The Dissertation Committee for Won Soo Kim
certifies that this is the approved version of the following dissertation:

Improving the Performance of Wireless Networks using Frame Aggregation and Rate Adaptation

Committee:

Scott M. Nettles, Supervisor

Gustavo de Veciana

Robert W. Heath, Jr.

Christine Julien

Lili Qiu

Improving the Performance of Wireless Networks using Frame Aggregation and Rate Adaptation

by

Won Soo Kim, B.E.; M.E.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2010

To my family

for their love and support

Acknowledgments

First of all, I would like to thank my supervisor, Dr. Scott Nettles, for his guidance and support. He guided me throughout my research at the University of Texas at Austin with remarkable insights and profound knowledge, and also encouraged me by showing me a great role model as a researcher, a teacher and also a human being.

I am indebted to Dr. Gustavo de Veciana for all his supportive guidance and advice on my research, to Dr. Robert Heath for all his advice and support, to Dr. Christine Julien for her kind encouragement and support, and to Dr. Lili Qiu for her helpful direction.

I would also like to thank colleagues of my research group, Soon-Hyeok Choi, Gibeom Kim, Ketan Mandke, Robert Grant, and Owais Khan for their helpful discussions and enjoyable working with them.

I would also like to thank my friends I met here in Austin, Byungchul Jang, Hyunsoo Park, Yongsang Kim, Taesoo Jun, Goo Jun, Hongseok Kim, Chan-Byoung Chae, Yuchul Kim, Junsung Yang, Junsung Park, Junghwan Han, Yonghyun Kim, Jaewook Lee, Kihyuk Han, Jin Heo, Jeongho Jo, Jaeyong Jeong, Kyoungeun Kwon, Sejun Yang, Inwook Kong, Sanghyun Chi, Yeojun Kim, Cheolhee Park, Junsoo Lee, Junsoo Kim, Dam Sunwoo, Hongjoong Shin, Jinkyu Lee, Jisun Park, Haewoon Nam, Hoojin Lee, Eunho Choi, Jery Xuan, Namhyun Um, Youngchun Kim, Yongseop In, Youngjae Yoo, Taehwan Choi, Hanhee Song, Jinmoo Lee, and Agisilaos Ziotopoulos for their friendship. My years in Austin were pleasant and enjoyable because of

them.

Finally, I would like to dedicate this dissertation to my wife Eunhye Kim, who has shared life and always supported me with love and encouragement, to my baby to be born in March 2011, and to my parents and parents-in-law for their love, support, and encouragement. Without them, it would have been impossible to accomplish this.

Won Soo Kim

The University of Texas at Austin
December 2010

Improving the Performance of Wireless Networks using Frame Aggregation and Rate Adaptation

Publication No. _____

Won Soo Kim, Ph.D.
The University of Texas at Austin, 2010

Supervisor: Scott M. Nettles

As the data rates supported by the physical layer increase, overheads increasingly dominate the throughput of wireless networks. A promising approach for reducing overheads is to group a number of frames together into one transmission. This can reduce the impact of overheads by sharing headers and the time spent waiting to gain access to the transmission floor. Traditional aggregation schemes require that frames that are aggregated all be destined to the same receiver. These approaches neglect the fact that transmissions are broadcast and a single transmission will potentially be received by many receivers. Thus, by taking advantage of the broadcast nature of wireless transmissions, overheads can be amortized over more data and achieve

more performance gain.

To show this, we design a series of MAC-based aggregation protocols that take advantage of rate adaptation and the broadcast nature of wireless transmissions. We first show the design of a system that can aggregate both unicast and broadcast frames. Further, the system can classify TCP ACK segments so that they can be aggregated with TCP data flowing in the opposite direction. Second, we develop a rate-adaptive frame aggregation scheme that allows us to find the best aggregation size by tracking the size based on received data frames and the data rate chosen by rate adaptation. Third, we develop a multi-destination frame aggregation scheme to aggregate broadcast frames and unicast frames that are destined for different receivers using delayed ACKs. Using a delayed ACK scheme allows multiple receivers to control transmission time of the ACKs. Finally, we extend multi-destination rate-adaptive frame aggregation to allow piggybacking of various types of metadata with user packets. This promises to lower the impact of metadata-based control protocols on data transport.

A novel aspect of our work is that we implement and validate the designs not through simulation, but rather using our wireless node prototype, Hydra, which supports a high performance PHY based on 802.11n. To validate our designs, we conduct extensive experiments both on real and emulator-based channels and measure system performance.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Wireless networking, and IEEE 802.11 [1] in particular, has been extremely successful. That success continues to grow and results in a significant need for greater bandwidth. To meet this need, wireless networks are deploying sophisticated broadband physical (PHY) layer technologies. For example, 802.11a/g [2,3] uses orthogonal frequency division multiplexing (OFDM), while 802.11n [4] adds multiple-input multiple-output (MIMO) techniques.

These technologies allow the data portions of frames to be transmitted at high data rates. This decreases the time spent transmitting data, but does not generally decrease the time spent on a variety of overheads. These overheads are incurred on a per transmission basis and include the time spent waiting to gain access to the transmission floor, exchanging control frames required by the medium access control (MAC) protocol, and PHY header. The result is that these overheads begin to dominate performance even when the PHY is capable of high data rates. In general, this problem becomes more severe as rate increases because the time to transmit the data decreases, but the duration of many of the overheads does not. Further, the effect of these overheads is more significant for short frames, such as those typically used for control.

The PHYs used in wireless networking allow packets to be transmitted at a variety of rates by controlling modulation, coding rate, and the number of data streams. The rate that can be supported reliably depends on the quality of the wireless channel over which the packet is transmitted, and choosing the highest reliable rate can improve system performance. However, when a channel is good, this technique increases the relative impact of overheads, which results in limited performance improvement.

One approach to reducing the impact of these overheads is to group (or *aggregate*) multiple frames into one transmission. This reduces overhead by allowing several frames to share headers and the waiting time to gain access to the transmission floor. In addition, this reduces the total number of transmissions, resulting in less time waiting for the floor and transmitting control frames. For example, the IEEE 802.11n standard includes some frame aggregation schemes as part of its high-throughput MAC design [4]. However, most frame aggregation schemes, including the IEEE 802.11n standard, require that frames that are aggregated all be destined to the same receiver. This approach neglects the fact that transmissions are broadcast and a single transmission will potentially be received by many receivers. Thus, we can take advantage of the broadcast nature of wireless transmissions by grouping multiple frames that are destined to different receivers into a single transmission. This is important because aggregating more frames amortizes overheads over more data, which results in achieving more improvement.

When does frame aggregation become effective? A general observation is that grouping multiple frames is possible only when frames are queued for transmission. When there is limited queueing, the load on the network is low. In this case, the overheads that can be reduced by frame aggregation have a limited impact on performance. When there is significant queueing, the load on the network is high. In this case, the overheads that can be reduced by frame aggregation have a significant

impact on performance. It is exactly this case where frame aggregation is both needed and effective.

Because information in the wireless network is broadcast, there is a further opportunity that involves aggregating metadata with the frames. Metadata is information that is not part of the data that is actually being transported by the network but rather is other data which might be useful for some reason. For example, we will see a feedback-based rate adaptation protocol later where we carry rate information in addition to user data. Another example is that we can imagine a MAC protocol where we might broadcast a node's queue state. Another example is that some ad-hoc routing protocols broadcast flooding packets to discover or maintain a route. Because we can piggyback this information onto an existing transmission without incurring additional overhead for sending this information, frame aggregation gives us an approach for transmitting this information at very low cost.

In this study, we focus on improving network performance in 802.11-style networks. In these networks, we will see slow time-varying channels caused by people walking and faster time-varying channels caused by a moving vehicle (e.g., buses), and frequency selective channels caused by multi-path signals, which becomes more significant in a large open space and/or broadband wireless systems. In addition, we consider both the infrastructured and ad-hoc networks that 802.11 systems support.

1.1 Thesis Statement

Our thesis is:

> Each transmission in an 802.11-style network incurs significant overhead. By aggregating multiple data frames and other information into a single transmission, this overhead can be amortized over more data, resulting in improved performance. Further, taking advantage of the broadcast nature of wireless

transmissions can result in greater amortization, and thus reduce the impact of overhead.

1.2 Goals and Approaches

Our goal is to demonstrate our thesis by designing new aggregation techniques that take advantage of rate adaptation and the broadcast nature of wireless transmissions. We validate our designs by implementing them using our wireless node prototype, Hydra [5], conducting extensive experiments, and measuring system performance. There are four steps to achieve our goal. The work of each step is presented as an independent Chapter.

First, we demonstrate a system that can aggregate both unicast and broadcast frames. Further, the system can classify transmission control protocol (TCP) acknowledgement (ACK) segments so that they can be aggregated with TCP data flowing in the opposite direction. This approach improves channel utilization by aggregating standard broadcast and TCP ACK packets with unicast frames, which results in less time waiting for the floor and transmitting control frames, and reduced MAC and PHY header overhead. The measured performance is compared with the frame aggregation method adopted by the IEEE 802.11n standard. This method is only able to aggregate multiple frames that are destined to the same receiver.

Second, we design a rate-adaptive frame aggregation scheme that combines the above design with rate adaptation. To maximize throughput, our design allows us to find the best aggregation size by tracking the size based on received data frames and the data rate chosen by rate adaptation. This results in reducing overhead and the total number of transmissions even further. The measured performance is compared with the frame aggregation method adopted by the IEEE 802.11n standard, which uses a fixed aggregation size for all possible data rates.

Third, we extend our MAC design to aggregate unicast frames that are

destined to different receivers using a delayed ACK scheme. Aggregating multi-destination unicast frames is challenging because of the difficulty of controlling multiple link-level ACKs from multiple receivers. Using delayed ACKs addresses this problem because they allow multiple receivers to control the transmission time of the ACKs. The measured performance is compared with the frame aggregation method adopted by the IEEE 802.11n standard and our aggregation scheme using the special approach for TCP ACKs.

Finally, we further extend the multi-destination rate-adaptive frame aggregation to piggyback metadata with user packets. Many protocols exploit metadata to improve network performance. However, transmitting metadata incurs some cost and this limits performance improvement. The goal of this work is to develop a transmit mechanism for piggybacking such metadata and to study the impact of overhead of sending metadata on network performance. This promises to lower the impact of metadata-based control protocols on data transport. The measured performance is compared with the frame aggregation method not supporting metadata piggybacking.

1.3 Road Map

The remainder of this dissertation is organized as follows. In Chapter 2, we discuss the detailed motivation of our approach and present the background material that is necessary to understand the rest of this dissertation. In Chapter 3, we present a new frame aggregation scheme, which allows the MAC to aggregate standard broadcast and TCP ACK packets with unicast frames. In Chapter 4, we present a new MAC design that supports both the rate adaptation and frame aggregation methods. In Chapter 5, we extend our study to allow the design to aggregate broadcast frames and unicast frames that are destined for different receivers. In Chapter 6, we further extend our study to allow the design to piggyback various kinds of metadata. In

Chapter 7, we present future work. Finally in Chapter 8, we conclude with the contributions of our work. In Appendix A, B, C, and D, we present the details of implementation for our designs discussed in Chapter 3, 4, 5, and 6 respectively.

Chapter 2

Motivation and Background

We discuss the detailed motivation of our approach and present the background material that is needed to understand the rest of the dissertation. First, we discuss the overheads in wireless networks. In addition, we describe the impact of overheads and show some analysis of overhead as a function of packet size and data rate. Second, we present a basic overview of frame aggregation and show the potential impact of frame aggregation on throughput. In addition, we discuss existing frame aggregation schemes. Third, we present the details of Hydra, a wireless network node prototype, used to evaluate the designs described in the following Chapters. Fourth, we present experimental issues with Hydra. Finally, we present the basic idea of rate adaptation and then discuss some rate adaptation algorithms. Further, we present some experimental results for rate adaptation.

2.1 Overheads in Wireless Networks

Every transmission in a wireless network incurs some overhead. These overheads are necessary for successful transmission, but they limit the achievable throughput because they take some time. We present the sources of overhead incurred by

individual transmissions and analyze the impact of these overheads on performance.

2.1.1 Individual Transmissions

Each transmission incurs overhead that can be categorized into per-transmission and per-frame overheads. Per-transmission overheads originate in the PHY, and in the process of acquiring the transmission floor, while per-frame overheads originate in the MAC and higher layers.

Per-transmission overhead in the PHY arises because each transmission requires a PHY header for frame synchronization, channel estimation, and signalling decoding information. The header overhead for synchronization is used to detect the point where a packet begins as well as to correct for frequency offset. The overhead for channel estimation is used to estimate the amplitude and phase distortions of the wireless channel and communication system. Signalling information includes the packet size and data rate, which allows a receiver to decode the packet. In addition, OFDM systems use pilot tones to correct for residual frequency offsets. For example, the IEEE 802.11a system includes 2 OFDM symbols for synchronization and channel estimation, 1 OFDM symbol for signalling, and 4 samples on a per OFDM symbol for pilot tones. Each OFDM symbol consists of 80 physical samples and duration of each sample is $\frac{1}{20\ MHz}$ or 50 ns [2]. In general, these overheads are transmitted using a base rate and thus are not sensitive to data rate.

Floor acquisition overhead arises because transmissions are broadcast. Thus, only one sender can transmit data over a wireless channel at a time, and others must defer their own transmissions until they "acquire the [transmission] floor". This results in a waiting time to gain access to the channel. In IEEE 802.11 systems, this is done using carrier sense multiple access with collision avoidance (CSMA/CA). We focus on IEEE 802.11 system but many of the issues would be similar regardless of what kind of system is used. In general, the time needed for acquiring the floor

does not change even when the data rate increases.

The basic carrier sense mechanism introduces overhead at a number of points. First, a sender waits to begin its own transmission when it senses carrier. Second, senders use additional waiting time to decorrelate transmissions from other nodes' transmissions when the channel becomes idle by using a random backoff. The waiting time corresponds to the number of random backoff slots. The number of slots is determined by taking a random value from a uniform distribution over the range from 0 to the contention window size. Third, retransmission might be required because of a collision or because the channel cannot support the data rate. Retransmission makes per-transmission overhead worse because it requires additional time to retransmit the packet. In addition, retransmission causes an exponential increase of the contention window. This increases the probability that the retransmission uses a longer backoff, which results in more waiting time.

One way of lowering the cost of collisions is to exchange control frames before the data transmission so that we can reduce retransmission overhead. For example, the IEEE 802.11 MAC uses a collision avoidance mechanism with a request-to-send (RTS) and clear-to-send (CTS) exchange. This exchange negotiates clearing the floor around the receiver, and successful negotiation allows a sender to transmit a packet with a low probability of collision. However this requires additional time for the control frame exchange.

Per-frame overheads include the headers and checksums for the MAC and higher layers. The MAC requires a header that comprises MAC-level control information and addresses. The addresses specify the transmitter and intended receiver(s). In addition, the MAC appends a checksum to each frame to detect errors. The IEEE 802.11 MAC includes a header of 30 bytes and a 4-byte checksum [1]. The Internet protocol (IP) and TCP have additional headers. The IP header contains 20 bytes of packet information and addresses [6]. The TCP header includes

20 bytes of source and destination ports and some control information [6]. In addition, other higher-level protocols have their own headers, which further increases the per-frame overhead. Different from per-transmission overhead, the time needed for these overheads varies with data rate.

There have been many efforts focused on improving overheads at multiple layers [7–9]. For example, one approach is to reduce per-frame overhead by concatenating headers above the MAC layer [8, 9]. This dissertation focuses on improving overheads at the PHY and MAC layers. Thus we assume that other layers' overheads are fixed. Furthermore, we mainly focus on improving per-transmission overhead rather than per-frame overhead. This is because per-transmission overhead is mostly fixed in cost, while per-frame overhead varies with data rate. This implies that, for higher rates, per-transmission overhead becomes more significant relative to per-frame overhead.

2.1.2 Impact of Overhead

What is the impact of the per-transmission and per-frame overheads? We consider the relative overhead, which denotes the ratio of the amount of bandwidth used for overhead compared to that used for total transmission. The relative overhead is more significant for smaller sized packets. This is because sending a smaller sized packet decreases the time to transmit the data, but does not decrease the time used by most of the overheads. Using a higher data rate is similar to sending a smaller sized packet. This is because using higher rates decreases the time to transmit data and most per-frame overheads but does not decrease the time to transmit per-transmission overhead. Thus this also increases the relative overhead.

As an example, suppose that we have a maximum (2304B) MAC frame at the base data rate. The percentage of the bandwidth used by overhead is 5%. If we send a TCP ACK at the base rate, then the overhead increases to 44%. If we send

these frames at the maximum data rate, the overhead for the maximum frame size is 43% and for the TCP ACK is 91%. Thus, the impact of overhead becomes more significant at a smaller sized packet or higher data rate.

To gain further insight into this issue, we analyzed overhead as a function of data rate and packet size. For this analysis, we derived lower bounds for the overhead using two assumptions: first that there were no transmission errors due to the wireless channel or collisions, and second that we ignored carrier sensing. These assumptions imply that we calculated a lower bound on the floor acquisition time and thus the total transmission time is also a lower bound. However, the absolute time to transmit the MAC and PHY headers is not approximated. In a real system, we would expect there to be additional overhead of waiting for floor and retransmissions.

For each data rate and packet size, we calculated the time to transmit MAC and PHY headers and the lower bound of floor acquisition time. Then, we derived the percentage overhead by dividing the duration of each overhead by total transmission time. Floor acquisition time consists of the interframe time, the expectation value of the backoff time corresponding to initial contention window, and the transmission time for the RTS/CTS exchange. The time needed for the RTS/CTS exchange is 81% of the floor acquisition time. One way of tackling this overhead could be to decide when we need the RTS/CTS exchange or not. Based on IEEE 802.11n [4], the MAC and PHY headers are set to 40 Bytes and 7 OFDM symbols respectively.

First, we consider the impact of data rate on overhead. Figure 2.1 shows the overhead percentage as a function of PHY data rate for a packet size of 1 Kbyte. The X-axis shows the PHY data rate in Mbps and the Y-axis shows the percentage of time used by overhead. This result shows that the overheads for both floor acquisition and PHY header dominate as the PHY data rate increases. This is

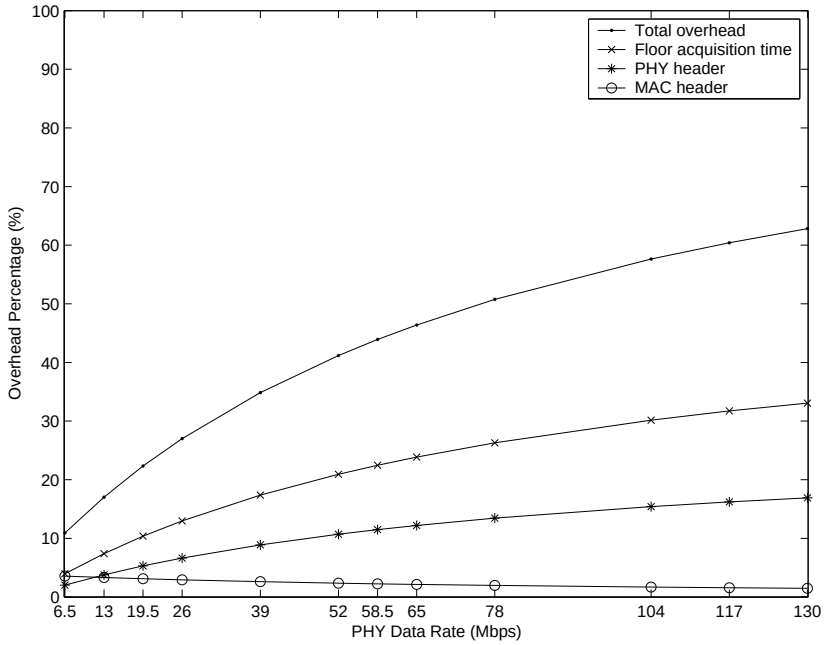


Figure 2.1: Overhead percentage vs. PHY data rate

because the floor acquisition time and PHY header overhead are not changed by the data rate and thus the impact of this overhead becomes more significant at higher rates. On the other hand, the MAC overhead becomes negligible as the rate increases. Further, this result shows that, for all rates, the floor acquisition time is the most important even though this is a lower bound. Thus we expect that this factor will become more significant as network load becomes higher. This is because high network load increases the time of waiting for floor and retransmissions.

Next, we consider the impact of packet size on overhead. Figure 2.2 shows overhead percentage as a function of packet size for the rate of 130 Mbps. This rate is the highest one that Hydra's current 2×2 multi-antenna system supports. The X-axis shows the packet size in bytes, and the Y-axis shows the percentage of time used by overheads. The X-axis ranges from 100B to 10KB. The result shows that the total overhead increases up to 94% as packet size decreases, and most of the

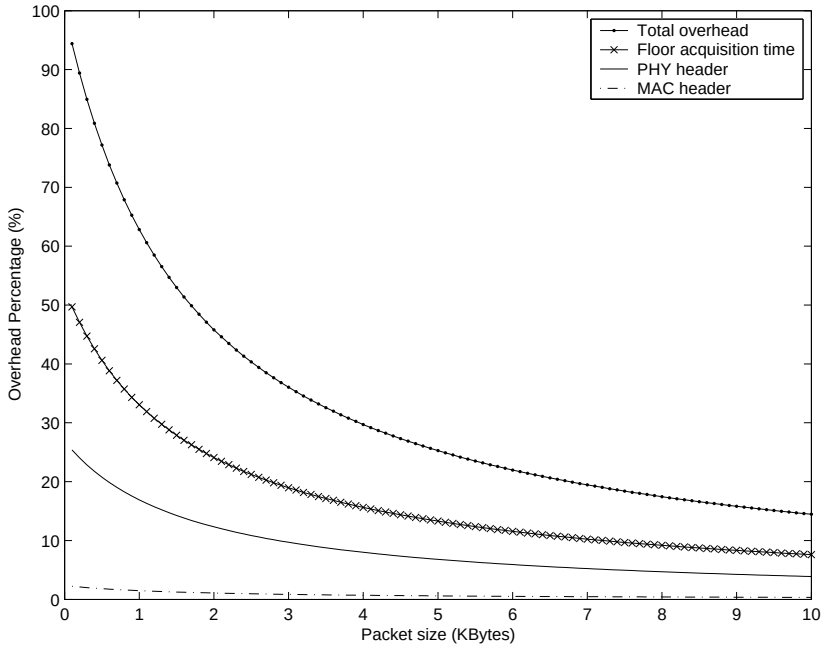


Figure 2.2: Overhead percentage vs. Packet size

overhead is incurred by floor acquisition and the PHY header. The total overhead for maximum Ethernet packet (1500B) is 53% and for a typical control packet of TCP ACK increases to 91%.

2.2 Frame Aggregation

Our overhead analysis shows that the per-transmission overheads have a great impact on performance. This impact becomes more significant as the PHY data rate increases or the packet size decreases. Obviously, whenever possible, it is a good idea to use a higher data rate for data transmission. However, for higher rates, the relative overhead shows a significant increase, limiting performance improvement. One approach to reduce the relative overhead is to use a lower data rate, but this is obviously not desirable. Thus the only way to reduce overhead is to make the frame bigger. In particular, there are a lot of control packets in real networks and

thus this is important because those are small. This can be achieved by grouping several frames together into a single transmission. This is called *frame aggregation*.

In this Section, we start by presenting the basic idea of frame aggregation. Second, we show a basic analysis of the achievable throughput of frame aggregation. Third, we present the IEEE 802.11n MAC, which adopts various frame aggregation methods. Finally, we discuss others' work related to frame aggregation.

2.2.1 Basic Idea

One approach to reducing these overheads and thus achieving the potential performance gains offered by modern PHYs is to group (or *aggregate*) several frames together into one transmission. This has two benefits: one, it reduces the total number of transmissions, resulting in less time waiting for the floor and transmitting control frames, and two, it reduces header overhead by allowing several frames to share headers. As an example of this approach, the IEEE 802.11n standard includes several frame aggregation schemes to support high data rates as part of its high throughput MAC design [4].

Basically there are two approaches to aggregating MAC frames. The first approach is to group multiple frames into a single transmission, each of which has its own MAC header and checksum. This approach does not save the per-frame overhead though it can save the per-transmission overhead. However, the advantage of this technique is that it can use a partial ACK scheme by detecting partial drops for individual subframes. A partial drop means that a received frame contains some erroneous subframes. The other approach is to aggregate multiple frames by concatenating user data using a single MAC header and checksum. The advantage of this technique is that this saves both the per-frame and per-transmission overheads. However, when partial drops happen, this approach cannot observe the partial drops for individual subframes and thus all the subframes are discarded.

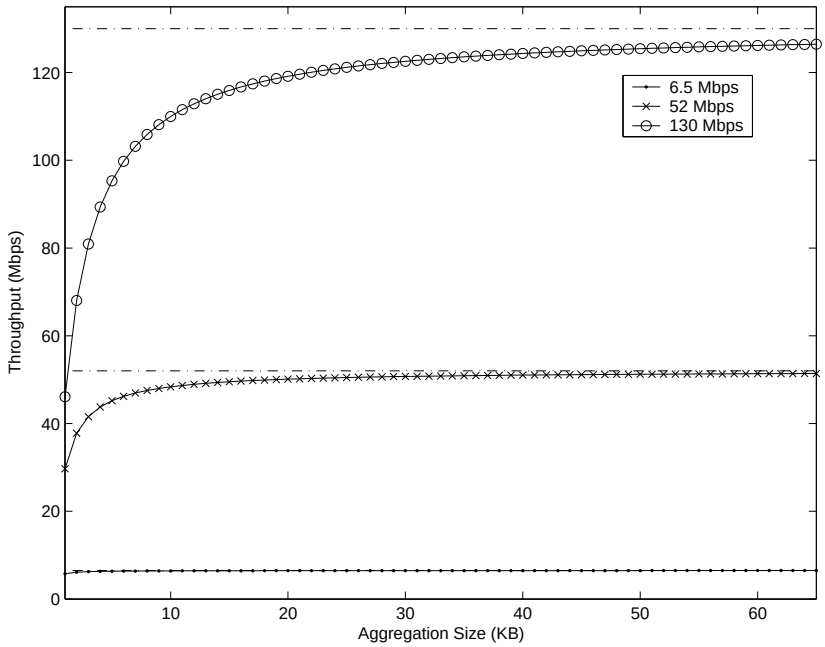


Figure 2.3: Throughput with a fixed data rate

2.2.2 Throughput Analysis

The goal of our throughput analysis is to derive the achievable throughput when frame aggregation is used. We analyzed the throughput with the same assumption and parameters as described in Section 2.1.2.

Figure 2.3 shows throughput as a function of aggregation size for the PHY rate of 6.5, 52, and 130 Mbps. We chose these PHY rates as the lowest, medium, and highest ones that Hydra's current 2×2 multi-antenna system supports. The aggregation size defines the total size of subframes that are aggregated. The X-axis shows the aggregation size in KB and the Y-axis shows throughput in Mbps. The dotted lines show asymptotic lines for the data rates of 6.5, 52, and 130 Mbps. We set the individual packet size to 1KB. We performed these analysis with the aggregation sizes up to 65KB.

We observe that, for all PHY rates, the throughput increases rapidly and, at

a point, it converges to the PHY data rate. As the PHY data rate increases, the slope of the throughput increases and the convergence point is shifted to the right side. This is because, at higher rates, overhead becomes more significant, and aggregating multiple frames reduces the overhead significantly. Thus the aggregation method is more effective to higher rates.

2.2.3 The IEEE 802.11n MAC

To improve throughput, the IEEE 802.11n [4] high-throughput standard adopts two approaches to frame aggregation: the aggregated MAC service data unit (A-MSDU), and the aggregated MAC protocol data unit (A-MPDU) [10]. A-MSDU aggregates packets from the upper layer and adds a single MAC header and checksum. This scheme is effective when the MAC aggregates many small user packets such as TCP ACKs or other control-oriented data. A-MPDU concatenates normal 802.11 MAC frames each having its own MAC header and checksum. Each of these subframes is separated by a MAC delimiter, which includes a length, checksum, and delimiter signature. The MAC delimiter allows a receiver to robustly separate each subframe, even in the case where some errors occur in the individual subframe. This approach has more overhead than the first, but supports a block ACK scheme. Block ACKs allow each subframe to be acknowledged separately, thus allowing retransmission of only the subframes in error. This approach will have an advantage with high error rates. Kim *et al.* [11] evaluated the throughput of an early variant of 802.11n frame aggregation as a function of payload size and physical data rate.

The 802.11n MAC also specifies a bi-directional data transfer method that can reduce floor acquisition overhead [10]. It is particularly useful for reducing the overhead of a bi-directional stream of TCP data and ACKs. When a node transmits a frame, instead of relinquishing the floor when the transmission completes, the node can grant the receiver permission for a reverse direction transmission destined for

the original transmitter. This approach allows both TCP data and ACKs to be transmitted in turn. This saves a floor acquisition time and the time to exchange a RTS and CTS if they are being used. However, this method does not reduce the MAC and PHY header overheads or the cost of link-level ACKs.

2.2.4 Related Work

Here we describe previous efforts to improve throughput using frame aggregation. Sadeghi *et al.* [12] proposed the opportunistic auto-rate (OAR) method, which uses frame aggregation to take advantage of favorable channel conditions. When the underlying rate adaptation algorithm shows that a frame can be sent at higher than the base-rate, the MAC attempts to aggregate frames so that the time spent sending the frame at the higher rate equals the time that would be the same as the time to send a single frame at base-rate. This preserves the basic fairness capabilities of the 802.11 MAC while taking advantage of higher rates and the overhead reduction of frame aggregation.

Skordoulis *et al.* [13] proposed a two-level frame aggregation scheme that mixes 802.11n's two aggregation methods. In the first stage, the MAC aggregates user packets from the upper layer into an A-MSDU with a MAC header and checksum. Then, a series of these A-MSDUs are concatenated into an A-MPDU with each A-MSDU separated by a MAC delimiter. This scheme increases the maximum aggregation size compared to using A-MSDUs and reduces MAC header overheads compared to using A-MPDUs. It allows the block ACK scheme to be applied to the A-MSDUs. The authors compared the throughput of their scheme with that of IEEE 802.11n frame aggregation schemes as a function of offered load.

Kim *et al.* [14] proposed a multi-layer scheme that provides aggregation at both the MAC and PHY layers. The MAC aggregates multiple MAC frames into an A-MPDU, and then the PHY aggregates a series of A-MPDUs into a single phys-

ical frame. Within the physical frame, an additional physical delimiter precedes each of the A-MPDUs. The physical delimiter contains modulation and coding information for each A-MPDU, and thus allows each A-MPDU to be transmitted at a different rate. Unlike the other existing approaches, this scheme also supports multi-destination aggregation because each A-MPDU can be addressed to a different destination. To facilitate this, the protocol employs a polling scheme so that frames sent to different destinations can be acknowledged. This scheme allows aggregation of multi-destination frames in the PHY. This saves floor acquisition time but cannot reduce overhead of physical headers. In addition, this scheme supports multi-destination aggregation in infrastructure-based networks. In contrast, our designs, discussed in the following Chapters, perform aggregation at the MAC layer and thus can reduce most of the per-transmission overhead. Further, our designs can support multiple destination aggregation even in an infrastructureless network.

2.3 Hydra

All the experiments presented in the following Chapters are performed using a wireless network prototype, Hydra [1] [5]. Hydra was designed to allow both the PHY and the MAC to be flexible and easy to modify. This design allows us to experiment with cross-layer designs on realistic hardware and real RF channels rather than just in simulations. Hydra has been presented in some detail in [17], and an overview of a variety of PHY issues that arise when using Hydra (and when experimenting with PHYs, MACs, and cross-layer design) appears in [18]. Here we present a brief overview of Hydra.

Figure 2.4 presents a block diagram of the main components of Hydra including the RF front-end, the PHY, and the MAC. The programmable RF front-end is the universal software radio peripheral (USRP) [19], which interfaces to the general

[1] This discussion is substantially taken from [15, 16]

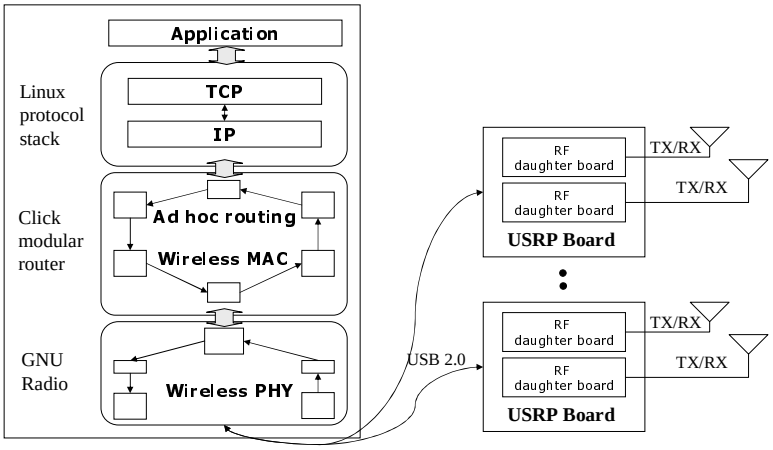


Figure 2.4: Block diagram of a Hydra node

purpose host through a USB 2.0 connection. The diagram shows several USRPs with multiple antennas, but here we use a single USRP with two antennas. All other aspects of Hydra, the PHY, MAC, and higher layers, run on a general purpose processor (GPP) running Linux. The PHY is written in C++ using the GNU Radio framework [20]. The MAC is also written in C++, but using the Click programmable router framework [21]. Click also provides ad-hoc routing and interfaces to the standard Linux stack. Implementing the PHY and MAC in C++ using general frameworks greatly eases the task of creating working cross-layer prototypes.

2.3.1 PHY

The Hydra PHY implements a MIMO-OFDM transceiver that is based on 802.11n [4]. Data is carried on 52 subcarriers that are modulated using BPSK, QPSK, 16-QAM, or 64-QAM. Forward error correction codes (punctured convolutional codes) are used with coding rates of 1/2, 2/3, 3/4, or 5/6. Our current 2×2 multiple-antenna system is able to support both single-stream and double-stream transmissions by using beamforming, spatial multiplexing, and cyclic delay diversity. The Hydra PHY can thus support various rates through a combination of modulation scheme, cod-

Table 2.1: Hydra PHY details

System Bandwidth	2 MHz^*
Center Frequency	$2.4 - 2.5$ GHz
Maximum TX Power	10 mW
Modulation	BPSK, QPSK, 16-QAM, 64-QAM
Coding	Bit-Interleaved Binary Convolutional
SISO Data Rates	0.65^*, 1.30^*, 1.95^*, 2.60^*, 3.90^*, 5.20^*, 5.85^*, 6.50^* $Mbps$
MIMO Data Rates	$2\times$, $3\times^+$, and $4\times^+$ SISO Data Rates
Diversity Schemes	Cyclic Delay Diversity, Space-time Coding, Spatial Mapping, Beamforming

* indicates non-standard values
$^+$ indicates capabilities in development

ing rate, and the number of data streams. In particular, the rates supported by the prototype are (in Mbps) 0.65, 1.3, 1.95, 2.6, 3.9, 5.2, 5.85, and 6.5 for single-stream mode and 1.3, 2.6, 3.9, 5.2, 7.8, 10.4, 11.7, and 13.0 for double-stream (spatial multiplexing) mode. Hydra's data rates are limited due to the bandwidth of the USB bus and processing delay created by the software implementation of the PHY. Thus the prototype supports physical layer data rates 10 times less than the actual data rates defined in the IEEE 802.11n standard. Table 2.1 summarizes the features of the Hydra's physical layer.

2.3.2 MAC

The MAC is written in C++ using the Click modular router framework [21]. This software framework, developed at MIT, runs on a general purpose processor and was originally created for building flexible and high performance routers. Similar to GNU radio, Click allows users to build packet processing elements in C++ and connects them using its own glue language.

The Hydra MAC follows the IEEE 802.11 MAC standard for distributed coordination function (DCF) with a RTS/CTS exchange. In addition, Hydra supports an

explicit feedback scheme using the RTS/CTS exchange and rate adaptation schemes including receiver-based auto rate (RBAR) and auto rate fallback (ARF) [22, 23].

2.4 Experimental Issues for Hydra

Since Hydra is implemented modularly in software and can be run on laptops, we can easily configure the system for different experiments. Many of the experiments presented in the following Chapters are run over real wireless channels using the USRP RF frontend. However, it can be difficult to create a variety of real channels, and achieving reproducibility in real channels is challenging. One advantage of Hydra's software implementation is that it is easy to create an emulator that processes the baseband output of the Hydra transmitter and then sends it to a Hydra receiver. Here, we discuss the setup for both our real and emulated channel experiments.

2.4.1 Real Channels

For our experiments over real channels, we use two or more laptops running the Hydra transceiver. Each transceiver is connected to a USRP with daughter cards operating in the 2.4 GHz band. Two antennas are connected to daughter cards, which allows us to do multi-antenna experiments.

All our experiments are set up in a small lab space. We used directional antennas that allow us to create various kinds of spatial correlations and fading. Changing the angles of the antennas allows us to change the relative power of the line-of-sight (LOS) and non-LOS signals in the channel, which impacts the Demmel condition number. We used the Demmel condition number as an indicator of the spatial structure of MIMO channels [24]. The distance between the nodes is such that, given the power constraints of the USRPs, we can explore a range of signal-to-noise ratios (SNRs) that allows us to adapt over the full set of rates. We vary the average SNR in these experiments by changing the transmission power.

Our experiments are done with stationary antennas. For these experiments, we calibrated the channel to find angles for the antennas that have good Demmel condition numbers. Fixing the angles allows us to have a relatively static spatial correlation. In addition, to create real channels that vary significantly in the time and spatial domains, our current best solution involves an oscillating metal fan along with one antenna mounted on each of the left and right sides of the fan.

2.4.2 Emulated Channels

The difficulty in creating and controlling real channels means that channel emulation is also important. Our experimental setup for an emulated channel consists of three GPP machines, two acting as Hydra nodes and one running the emulator code.

The channel emulator allows us to experiment with a variety of channel models and impairments, while still using the PHY and network stack that is part of a working system and that we use for experiments over real channels. This gives us the control of a simulation, but with greater fidelity to a working system.

For spatially uncorrelated time-varying multi-antenna channels, we used Jakes channel model [25], which is a model for rayleigh fading channel that includes Doppler effects. The Doppler effects are parameterized by normalized Doppler, $f_d T_s$, where f_d denotes Doppler frequency and T_s stands for the sample duration. For the following emulator-based experiments, T_s is fixed at $\frac{1}{2\ MHz}$. For the multi-antenna systems we generated independent identically distributed (i.i.d.) time-varying single antenna channels up to the number of multi-antenna channel elements. We assume that the channel can be changed within a single transmission based on channel coherence time, which approximates the inverse of the Doppler frequency [26].

To add frequency selectivity to the channel, we used the power-delay profile having exponential distribution [26]. In addition, to controlling average SNR, we changed the noise power in the additive white Gaussian noise (AWGN) channel

model.

2.4.3 Experimental Details

For our real experiments, we used a transmission power of 7.7 mW, and the spacing between nodes is roughly 3 meters. We did experiments at all possible rates for the single stream mode and the spatial multiplexing mode.

To create user datagram protocol (UDP) traffic, we used an application that simply sent UDP packets at a controllable rate. The size of a UDP packet is 1KB, which results in 1136B MAC frames. For TCP experiments, we used a one-way file transfer with a file size of 4 Mbytes. The maximum segment size for TCP is set to 1357 bytes, resulting in a MAC frame of 1464 bytes. TCP ACKs had a MAC frame size of 160 bytes.

Hydra's receiver PHY, implemented by software, takes a significant amount of time to process a packet, and that time is dependent on packet length. Thus, Hydra uses spacing between packets that are longer relative to the packet transmission times than for the 802.11n standard. Our goal is to provide insights about systems with more typical, hardware-based PHYs, and so we have processed the throughput results presented in the following Chapters so that they reflect the interframe spacing defined in the IEEE 802.11n standard [4]. This aspect of Hydra is discussed further in [18].

2.5 An Experimental Evaluation of Rate Adaptation

Eventually, our design will do frame aggregation in conjunction with rate adaptation. Thus, we need some basic understanding of rate adaptation. For this purpose, we first overview rate adaptation and then show some experimental results to evaluate some rate adaptation algorithms. This discussion is based on [16].

2.5.1 Rate Adaptation

Rate adaptation is a technique of controlling the transmission rate of packets by tracking wireless channels. This improves performance by decreasing packet transmission time. We present the basic idea of rate adaptation and then we discuss some rate adaptation algorithms.

Basic Idea

The goal of rate adaptation is to balance data transmission rate with the rate of packet failure. Choosing too low a rate will cause transmissions to take longer than needed, while choosing too high a rate causes the transmission to fail. For the systems considered here, three factors determine the rate: how the data is modulated, the amount of redundancy in the error correcting code, and the number of spatial multiplexing streams. If we fix the degree of spatial multiplexing, then for the systems under study here, the probability that a given modulation and coding rate will result in a successful transmission is determined primarily by the SNR at the receiver, which is in turn a function of the transmit power and the channel between the transmitter and the receiver. This is true for frequency flat fading channels. Changes in the channel mean that the best rate to use for transmission will change, and any approach to rate adaptation must track such changes to deliver the best rate.

The situation becomes more complex when there is a need to adapt between diversity and spatial multiplex modes in multi-antenna systems. Spatial multiplexing algorithms only perform well in certain types of wireless channels with *good* spatial structure. In other words, increasing the number of data streams in a channel with a *bad* spatial structure will significantly reduce reliability and decrease throughput. The Demmel condition number is a good indicator of the spatial structure of MIMO channels [24]. Thus what rate (including the degree of spatial multiplexing)

is best is a function of both the SNR and the Demmel condition number, both of which vary depending on the channel. Actually, the situation is more complex than this, and the Demmel condition number is most useful for frequency flat fading channels [27], such as those found in relatively narrowband systems such as Hydra.

Rate Adaptation Algorithms

Many rate adaptation protocols use implicit feedback to track the changing channel and adjust the rate. ARF [23] is typical of such algorithms. In ARF, ACKs are used as implicit feedback. If an ACK is not received, the current rate is assumed to be too high and the rate is lowered. If an ACK is received, the current rate is known to be equal to or lower than the ideal rate, and if enough ACKs are received in a row, the protocol increases the rate to see if a higher rate is feasible. ARF raises the rate if 10 consecutive ACKs are received and lowers it if there are two consecutive ACK failures.

Advantages of using ARF include that no information is needed from the PHY and packet formats are not impacted. However, ARF has the disadvantage that it can only adapt to changes in the channel that take place on time scales greater than a few packet exchange times. Also, even for an unchanging channel, it occasionally raises the rate to see if a higher rate can be supported, resulting in packet loss. Finally, ARF assumes that failure to receive an ACK is due to the rate being to high, even though the failure may have some other cause, such as a collision.

Another class of protocols uses explicit feedback to track the channel. A good example is RBAR [22], which uses the RTS preceding the data transmission to measure the SNR. The receiver then determines the best rate based on this SNR and returns this rate to the transmitter, which uses it to transmit the data. Although there are a multitude of feedback algorithms, RBAR captures the basic architecture.

An advantage of RBAR is it can adapt to channels that change at a rate faster than tens of packet times, and it does not need to probe when the channel does not change. One limitation is that it may not work well for channels that change so fast that the SNR for the RTS is different than for the packet. However, the main disadvantage is that it requires a potentially expensive RTS/CTS handshake to measure and feedback the channel related information, as well as additional space in the CTS for the feedback information, which impacts the packet format. Avoiding this overhead can improve network performance by saving the time to transmit control packets but may cause the loss of the advantage of having timely feedback.

2.5.2 Experimental Evaluation

We evaluated rate adaptation algorithms by performing a series of rate control experiments based on transmission over both real and emulator channels. Our experiments include measurements for single antenna systems and two antenna systems using a single or spatial multiplexing streams. We studied rate adaptation algorithms using both explicit and implicit feedback from the receiver.

Here, we present the experimental calibration of rate transition points for RBAR and, for further discussion, we show an experimental result to compare the throughput of RBAR with that of ARF for single stream rates. All other experimental results based on real and emulator channels have been discussed in [16].

Calibrating the RBAR transition points

Any RBAR implementation will need to make a choice about when to transition from one rate to another. As with Holland [22], we use SNR as the criteria for channel quality. We choose the transition points by experimental calibration using the fixed rates supported by Hydra as a function of SNR. Separate calibration is needed for single antenna, single stream, and spatial multiplexing cases. Here, we

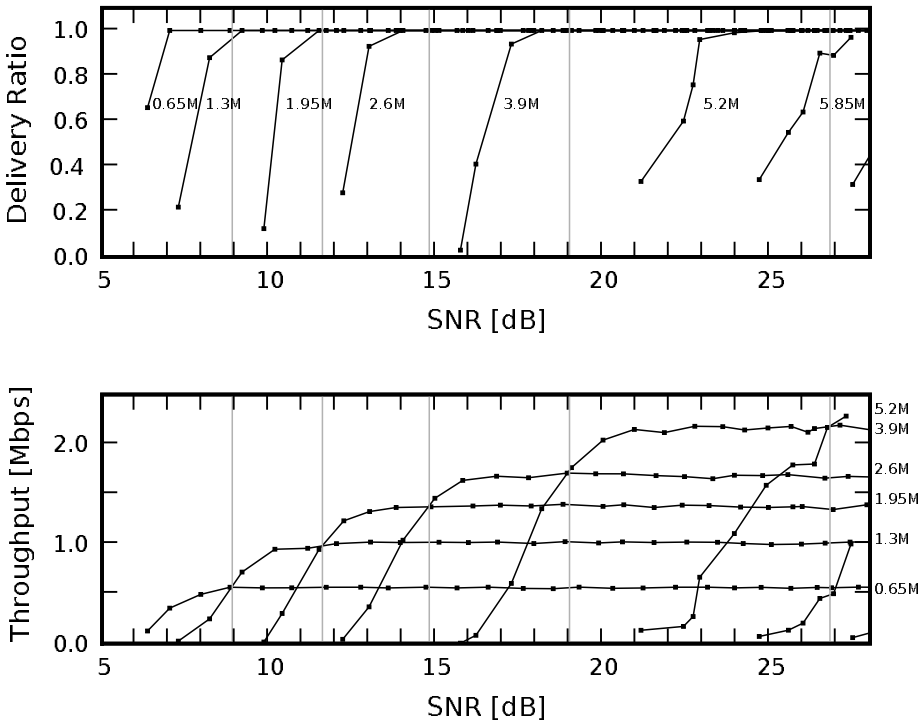


Figure 2.5: Delivery Ratio and Throughput vs. SNR for single antenna fixed data rates. Dotted vertical lines indicate rate transition points for RBAR.

present the results for single antenna and single stream; the calibration result for spatial multiplexing can be found in [16]. Note, because it depends on the channel, we cannot control the SNR directly. In these experiments, we control the transmit power and then measure the SNRs of the packets to find the required cutoffs.

Figures 2.5-2.6 show the results we used to choose our transition points. All graphs have SNR on the X-axis. These graphs show the results for all of the fixed rates supported by Hydra for that version of the system. For Figure 2.5 the Y-axis of the top graph shows the packet delivery ratio and of the bottom graph shows the throughput achieved in Mbps. The packet delivery ratio for the other measurements was essentially identical and so has not been presented. The vertical lines show the

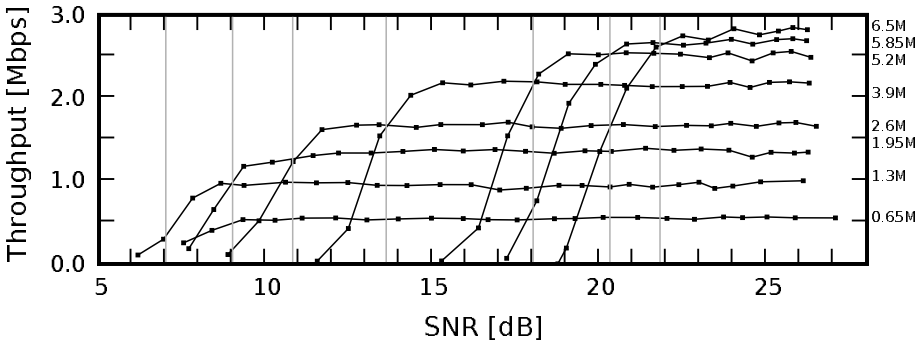


Figure 2.6: Throughput vs. SNR for single stream fixed data rates. Dotted vertical lines indicate rate transition points for RBAR.

transition points we chose for RBAR. Note that single stream supports higher rates than single antenna and the cutoffs chosen are lower than for single antenna. This reflects the improvement we get in the channel from using diversity.

One design detail is what criteria to use for choosing the rate transitions. We choose to transition from one rate to another when doing so would result in higher throughput. Thus the transition points are located at the points where the throughput of a higher rate crosses that of the best performing lower rate. As shown by the vertical lines, the transitions we use are (in dB): for single antenna, 8.9, 11.6, 14.8, 19.0, and 26.8; and for single stream, 7.0, 9.0, 10.8, 13.6, 18.0, 20.3, and 21.8. Note that the first cutoff for single stream is an estimate since the crossing point is not observed in the data.

RBAR and ARF with RTS/CTS

The goal of this rate adaptation experiment is to provide some understanding that rate adaptation, used for the design in Chapter 4, works in practice and shows real improvement of throughput. Figure 2.7 shows the experimental result comparing RBAR and ARF for single stream rates. The fixed single stream rate data is shown

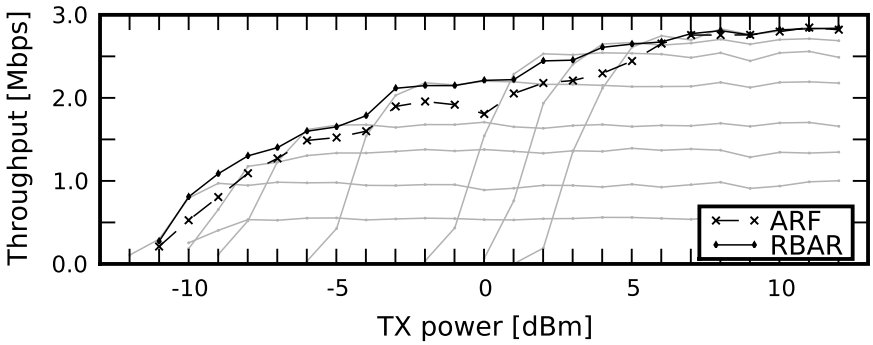


Figure 2.7: Throughput vs. TX power for single stream RBAR and ARF. Fixed rate lines are shown lightly for reference.

lightly for reference. Note here, we plot the data in terms of transmit power. Since transmit power is what is actually controlled at the transmitter, this result reflects the conditions actually experienced by ARF and RBAR.

We see that RBAR tracks the top of the curves formed by the fixed rate data relatively closely, indicating that RBAR is generally choosing the best rate possible. ARF also tracks the fixed rate data reasonably well, but not as closely as RBAR. ARF has some dips at about the points the fix rate results cross. We believe at these points, ARF is sometimes increasing its rate and incurring drops.

Chapter 3

Fixed-Rate Frame Aggregation

Most frame aggregation schemes require that frames that are aggregated all be destined to the same receiver. This approach neglects the fact that transmissions are broadcast and a single transmission will potentially be received by many receivers. A simple way of taking advantage of this is to aggregate broadcast frames along with a group of unicast frames all destined for one receiver. Because broadcast frames do not require acknowledgement, this can be done while still having a single ACK for the unicast frames.

We expect TCP traffic to be important for wireless networks, and it can also benefit from frame aggregation. The key observation is that TCP ACKs are small packets that flow in the opposite direction of the (typically) larger TCP data packets. Since TCP ACKs are cumulative and thus carry redundant information, they have lower reliability requirements than the data packets. We take advantage of this by treating TCP ACKs as if they were broadcast frames and do not require link-level acknowledgements. This allows the ACKs to be aggregated with TCP data traveling in the opposite direction, significantly reducing the cost of the TCP ACKs. By allowing the MAC to examine TCP headers, our design breaks layering abstractions and thus is a cross-layer algorithm.

Here, we present the design of a system that can aggregate both unicast and broadcast frames [1]. Further, the system can classify TCP ACK segments so that they can be aggregated with TCP data flowing in the opposite direction.

3.1 Related Work

There have been cross-layer approaches to improve TCP performance by piggybacking small TCP ACKs with link-level frames [28–31]. C. Parsa *et al.* [28] proposed the transport unaware link improvement protocol (TULIP) that provides a piggyback method which transfers a TCP ACK with a link-level frame. J. Tourrilhes [29] proposed PiggyData that is to transmit PiggyData ACK, a link-level acknowledgement with a flag indicating status of the transmit queue, as a response of TCP data. Setting the flag to one means that a TCP ACK is followed after short inter-frame space (SIFS) interval. Y. Xiao [30] suggested a piggyback mechanism that allows the MAC to piggyback a link-level acknowledgement with a TCP ACK in a single frame immediately after the node receives TCP data. Our scheme differs from these approaches in that these do not reduce the MAC and PHY header overheads or the cost of link-level ACKs.

L. Scalia *et al.* [31] suggested PiggyCode which allows the MAC to combine TCP data with TCP ACK by using network coding technology. This approach can reduce the MAC and PHY header overheads for TCP ACKs, similar to our approach. However, the PiggyCode requires additional headers to encode and decode packets, and it allows only packets of different types to be coded together, which restricts it to transmitting a single TCP data and TCP ACK at a time.

[1] This Chapter is substantially based on [15]

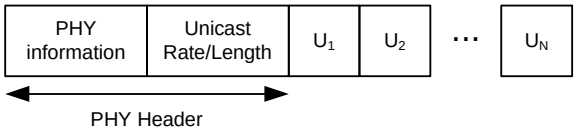


Figure 3.1: Unicast aggregation format

3.2 Design

Our design incrementally extends the familiar IEEE 802.11 DCF MAC protocol [1] to support frame aggregation. This addition requires modest changes to the PHY frame format, demonstrating one of the cross-layer aspects of our design. Extensions include support for unicast aggregation, broadcast aggregation, and link-level classification of TCP ACKs as broadcast frames. The implementation details of this design have been described in Appendix A.

3.2.1 Unicast Aggregation

Unicast aggregation follows the A-MPDU scheme adopted by the IEEE 802.11n standard as described in Section 2.2.3. Figure 3.1 is the format for supporting unicast aggregation, where U_N denotes the N-th unicast subframe. The aggregated frame consists of some PHY-oriented information, such as training sequences, the rate and length fields for the frame and then a series of unicast subframes, all bound for the same destination. Because all the unicast subframes are destined for the same node, a single ACK can be used to acknowledge all the subframes. Here, no changes are needed to the PHY.

3.2.2 Broadcast Aggregation

Broadcast frames are likely to be important in a multi-hop wireless network, especially for control protocols. For example, the dynamic source routing and ad-hoc on-demand distance vector routing protocols use broadcast frames for route discov-

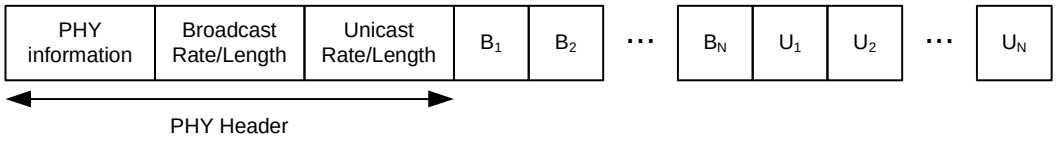


Figure 3.2: Broadcast aggregation format

ery and maintenance [32,33]. The broadcast nature of radio frequency transmissions means broadcast frames can not only be aggregated with each other, but also with unicast frames. This promises to significantly lower the impact of flooding-based control protocols on data transport.

Figure 3.2 shows how our broadcast aggregation format extends the basic unicast format, where B_N stands for the N-th broadcast subframe. Our design modifies the PHY header to add a rate and length field for the broadcast subframes. The rate information enables us to support different data rates for broadcast and unicast subframes. The length information allows our protocol to prepend a variable number of broadcast subframes to a variable number of unicast subframes, all within the same physical frame. Our design requires modifying the PHY header to include rate and length information to allow the receiving PHY to decode incoming streams. The broadcast subframes are not acknowledged and thus a single link-level ACK is still sufficient. In addition, depending on queue status, the broadcast aggregation scheme allows the MAC to aggregate broadcast or unicast frames only.

3.2.3 Treating TCP ACKs as Broadcasts

Many of the Internet's most important applications use TCP as the transport protocol. Thus, optimizing for TCP becomes important. Furthermore, TCP represents a general class of protocols that support reliable transmission. TCP relies upon a bi-directional traffic flow of TCP data and TCP ACKs. Because the ACKs are small, the impact of the fixed overhead becomes even more significant, making them

especially good candidates for aggregation.

TCP employs a cumulative acknowledgement mechanism. In this scheme, receipt of an ACK for packet P_i, where i is the sequence number of the packet, implies acknowledgement of all previous packets P_j, for $j \leq i$. Because of this redundancy, ACKs have less need for reliable transmission than data. Indeed, some TCP implementations intentionally drop some fraction of the ACKs to reduce protocol overhead [34]. This suggests that TCP ACKs could be transmitted without a link-level acknowledgement, in the same manner as broadcast frames.

Our design breaks layer boundaries in a novel way by classifying TCP ACKs as broadcast frames and then aggregating them in the same manner as frames with broadcast addresses. This can potentially cut the number of transmissions and thus floor acquisitions needed by a TCP flow in half as well as save the significant other overheads associated with transmitting the small TCP ACK frames.

TCP ACK aggregation does not require a new frame format. Instead, TCP ACKs are categorized as link-level broadcasts and transmitted in the broadcast portion of the frame. Although the TCP ACKs are transmitted as a broadcast subframe and thus do not generate link-level ACKs, they still have unicast MAC addresses. When a node receives a TCP ACK not addressed to it, it drops the packet, rather than passing it up the stack. This behavior is significant; if the packet was passed up the stack to the IP layer, it would attempt to deliver the packet, resulting in improper duplication of the TCP ACK. Thus, instead of broadcasting TCP ACKs to the entire network, as a typical broadcast packet, the ACKs are just broadcast within the range of the nodes along the TCP stream's path, just as they would be if the ACKs were sent as unicast packets.

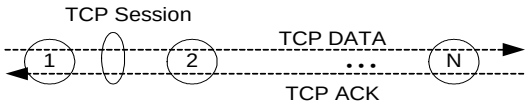


Figure 3.3: Linear topology with one TCP session

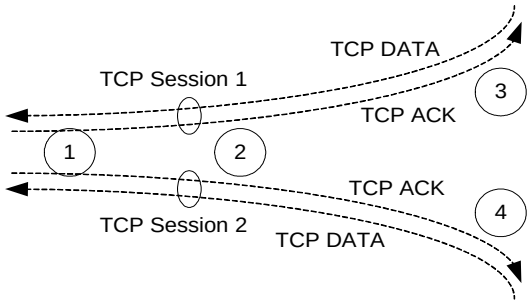


Figure 3.4: Star topology with two TCP sessions

3.3 Experimental Setup

For our experiments, we used the same experimental setup described in Section 2.4. In addition, we used the cyclic-diversity MIMO mode and thus transmitted a single data stream from our two antenna systems. We ultimately did experiments at 0.65, 1.30, 1.95, and 2.60 Mbps. Higher rates would have shown greater improvements for our techniques, but the rates used serve to validate our concept. Experiments for higher data rates will be presented in Section 5.2.

We used three different topologies: 2- and 3-hop linear topologies as shown in Figure 3.3; and a star topology as shown in Figure 3.4. Spacing between the nodes is roughly 2.5 meters. Although Click provides several ad-hoc routing protocols, because all of our nodes are within transmission range of the others, they would have not discovered any multi-hop routes. Instead we used static routing to force the topologies in the figures.

3.4 Experimental Results

We performed a series of experiments to study the impact of frame aggregation for unicast frames, broadcast frames, and TCP ACKs. To begin, we determined the aggregation size to be used for subsequent experiments. We then present performance results for each of our techniques. We conclude with a more detailed analysis that lets us gain some additional insight into how our protocols operate.

3.4.1 Maximum Aggregation Size

For our experiments, we needed to set a maximum aggregation size to be used in our further validation. Aggregating multiple frames improves the impact of overhead on data transport. Thus, we could use arbitrary aggregation sizes. However, as aggregation size grows, the improvement becomes very limited. Thus, it is important to bound the maximum aggregation size. Further, frame aggregation becomes more effective at higher rates, and thus we could bound the maximum aggregation size differently according to data rate. This motivates our design, discussed in Chapter 4, to extend the MAC to allow changing the aggregation size as a function of data rate.

For this design, we consider an aggregation scheme using a fixed aggregation size for all data rates. Thus, for the maximum aggregation size, we chose 5KB, which achieves 4% overhead improvement for the rate of 0.65 Mbps.

Table 3.1: 2-hop UDP throughput

Data Rate	No Aggregation	Unicast Aggregation	Difference
0.65 Mbps	0.253 Mbps	0.273 Mbps	7.9%
1.3 Mbps	0.430 Mbps	0.481 Mbps	11.9%

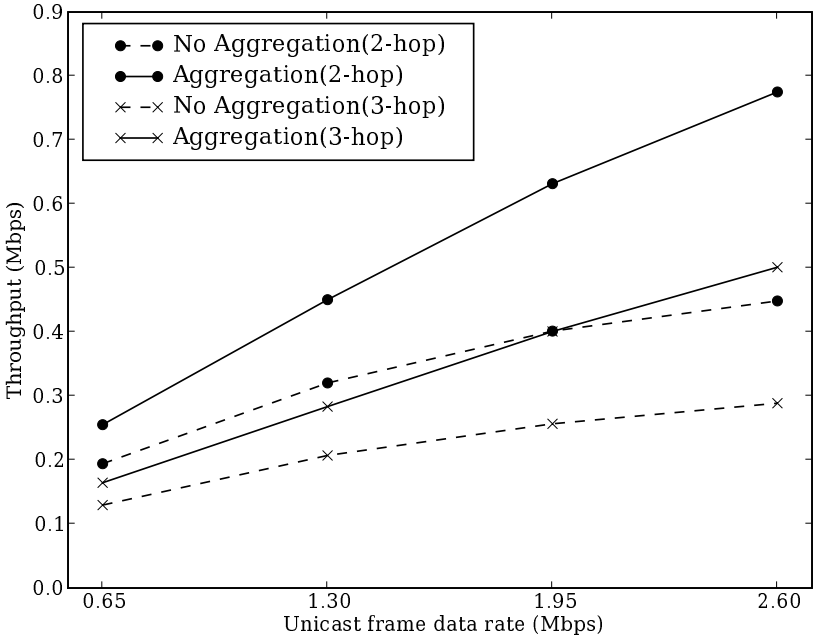


Figure 3.5: TCP unicast aggregation

3.4.2 Unicast Aggregation

We designed this experiment to study the impact of unicast aggregation. Because aggregation saves MAC and PHY overheads as well as transmissions, we expect that throughput will be enhanced and that the amount of enhancement will increase as rate increases.

For UDP traffic, we used our UDP application over a 2-hop network and fixed data rates of 0.65 Mbps and 1.3 Mbps. Table 3.1 presents throughput when the data interval is set to 3 seconds. We observed a significant improvement with aggregation and that, as expected, this improvement increases with rate.

For TCP traffic, we used a one-way file transfer over both 2- and 3-hop linear topologies. Figure 3.5 shows the experimental results for TCP as we varied the data rate. The X-axis shows the unicast data rate in Mbps and the Y-axis shows the end-

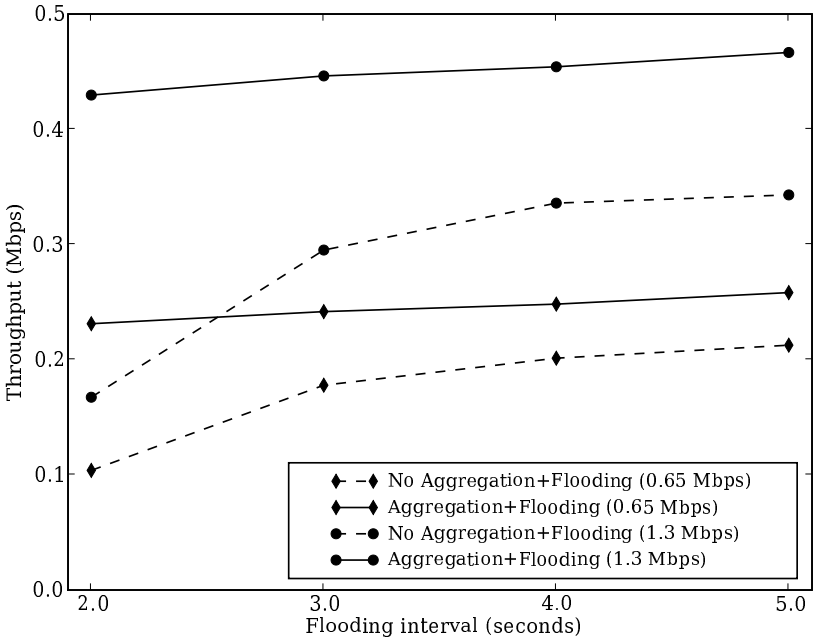


Figure 3.6: 2-hop UDP flooding

to-end throughput in Mbps. Figure 3.5 demonstrates that throughput is improved for both 2-hop and 3-hop networks when aggregation is applied. Again, as expected, the improvement increases as data rate increases.

3.4.3 Broadcast Aggregation

We designed this experiment to assess the impact of broadcast aggregation in the presence of flooding. By combining the flooding frames with unicast frames, we expect that the impact of flooding on performance will be greatly reduced. For these experiments both unicast and broadcast aggregation was enabled.

We used our UDP application over a 2-hop network with a data interval of 3 seconds, and fixed data rates of 0.65 Mbps and 1.3 Mbps. To simulate flooding, each node generated broadcast frames at a fixed rate, which we varied during the

experiments. Figure 3.6 shows the result of aggregation and no aggregation as a function of the flooding interval. The X-axis shows the interval of flooding frames in seconds and the Y-axis shows the end-to-end throughput in Mbps.

Our results show that for both data rates the performance gap between aggregation and no aggregation increases as the flooding interval decreases. A small flooding interval has a greater impact on the throughput when no aggregation is used, indicating that aggregation effectively reduces the overhead of flooding. We also observed the expected trend with increasing rate. If we compare this result with the unicast aggregation result when there is no flooding (Table 3.1), we find that for 0.65 Mbps with flooding at the 5 sec interval the throughput is 0.26 Mbps while without flooding it is 0.27 Mbps, while for 1.3 Mbps flooding has a throughput of 0.47 Mbps and without 0.48 Mbps. This degradation comes from two sources: first, only the throughput of the UDP application is considered; and second, as the flooding interval decreases, the number of aggregated frames containing only flooding frames increases.

3.4.4 TCP ACK Aggregation

We designed these experiments to study the effect of treating TCP ACKs as broadcast frames, thus enabling bi-directional aggregation. We study the impact of bi-directional aggregation in a variety of configurations. We start by measuring the throughput as a function of the unicast frame data rate, when the rate used by the broadcast subframes is fixed, followed by a measurement when the broadcast and unicast subframes use the same rate. We extend our experiments to 3-hop linear and star topologies to study the impact of hop count and network congestion. To study the impact of queueing delay and enhancements that might be gained by delaying frames to create greater opportunities for aggregation, we experimented with a version of our system that waited until it could aggregate 3 frames before transmission.

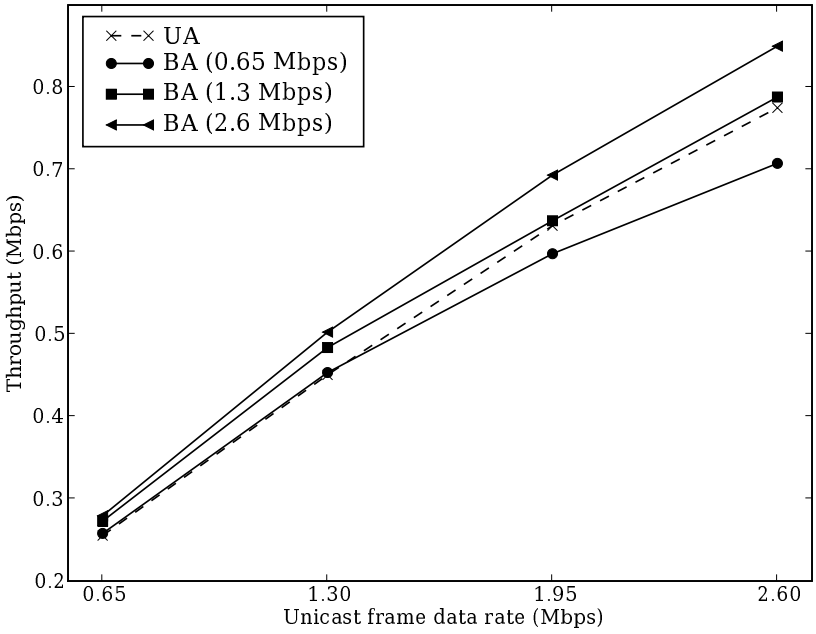


Figure 3.7: TCP ACK aggregation with a fixed broadcast rate

To separate the impact of backward and forward aggregation, we performed experiments for TCP ACK aggregation while disabling forward aggregation. Finally, we present some detailed analysis that lends greater insight into the behavior of our system.

All experiments in this Subsection use a one-way file transfer with a file size of 0.2 Mbytes to generate TCP traffic. We use the abbreviation BA to denote broadcast aggregation results, UA for only unicast aggregation, and NA for no aggregation.

2-hop Networks

We designed these experiments to compare TCP performance between BA and UA over a 2-hop network by studying throughput as a function of PHY data rate. In the first experiment, we varied the rate of the unicast part of the frame, but fixed

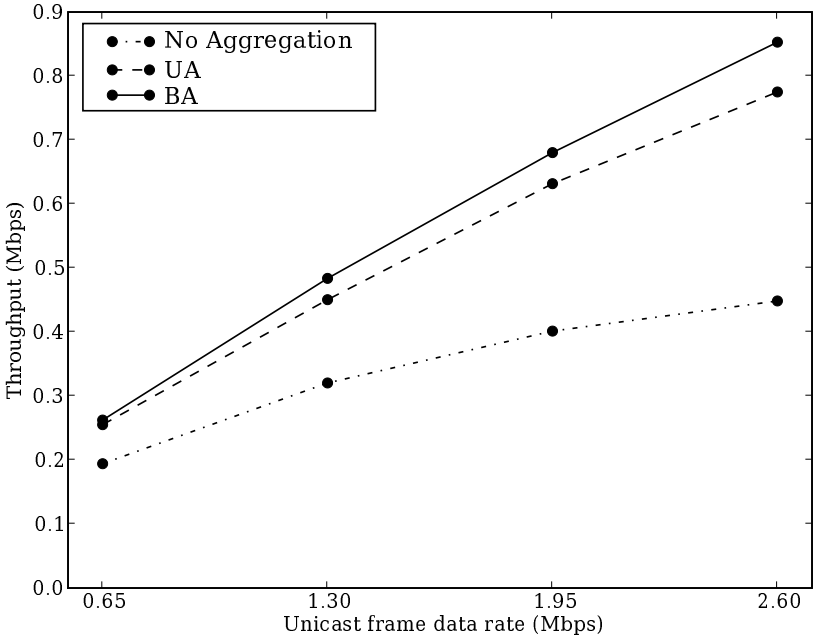


Figure 3.8: 2-hop TCP ACK aggregation

the rate of the broadcast part of the frame. In the second experiment, the broadcast and unicast subframes have the same rate.

Figure 3.7 shows the results for BA when using the fixed data rates of 0.65, 1.3, and 2.6 Mbps for the broadcast TCP ACKs, and for UA. The X-axis shows the unicast data rate measured in Mbps and the Y-axis shows the end-to-end throughput in Mbps. The value in parenthesis is the fixed rate used for the broadcast subframes.

When the MAC broadcasts TCP ACKs at 0.65 Mbps, we see that BA shows some improvement over UA at the unicast rate of 0.65 Mbps, but that as the unicast rate increases, the throughput of BA falls off. As the unicast rate goes up, the time spent transmitting the broadcast ACK at 0.65 Mbps increasingly dominates, causing this effect. When TCP ACKs are broadcast at 1.3 Mbps, we observe that BA outperforms UA up to the unicast rate of 1.3 Mbps, and afterwards BA achieves

performance similar to UA. BA using 1.3 Mbps becomes more effective because for high data rates the impact of aggregation becomes more significant. When TCP ACKs are broadcast at 2.6 Mbps, we observe that BA always outperforms UA over the range of rates used in our experiments, for the same reasons we have already mentioned.

Figure 3.8 shows similar results when the broadcast TCP ACKs use the same data rate as the unicast subframes. In this case BA always outperforms UA. Comparing BA with UA, the maximum throughput performance gap is 10%. (Also, improvements exist at 0.65 Mbps, but are hard to see at the scale used in the graph.) This is because BA can aggregate more subframes at the relay nodes, reducing both header overhead and the number of transmissions. We also include the no aggregation results to show that both BA and UA provide significant improvements over not doing any aggregation, as expected. Because it provided consistently better performance, we use this version with broadcast and unicast frames at the same rate for all subsequent experiments.

3-hop Linear and Star Networks

We designed these experiments to study the performance of our system with more complex topologies, such as when more relay nodes become involved or when the network becomes congested. We expected that BA would show an enhancement over UA due to increased hop count and that network congestion would increase the ratio of aggregation. In a star topology, we expected more congestion because the central node would be a bottleneck for the TCP streams. More congestion results in longer queues, which provide more chances for aggregation. In a 3-hop linear topology, we expected that adding a relay node would increase the number of nodes involved in aggregation of TCP data and ACKs, improving the aggregation ratio.

We used two topologies: a 3-hop linear topology (Figure 3.3) having more

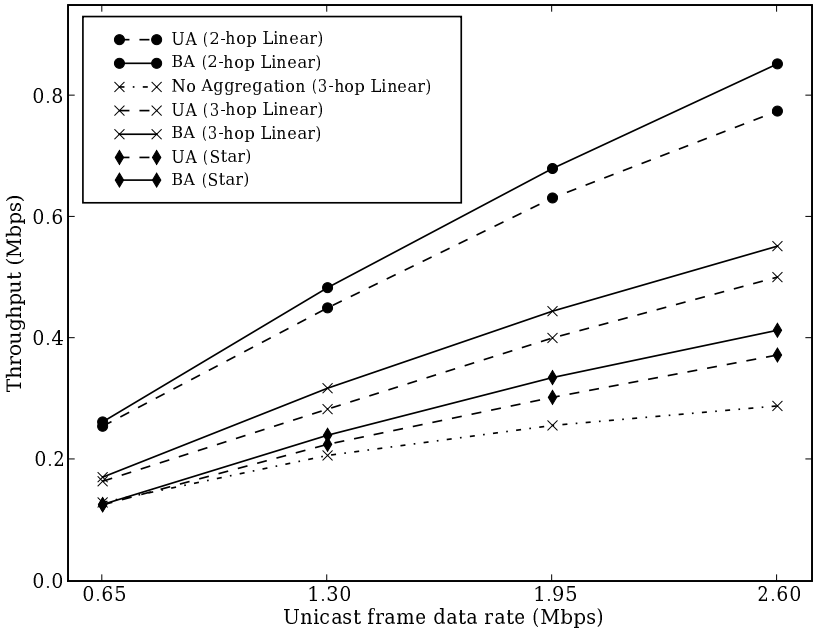


Figure 3.9: TCP over more complex topologies

hops and a star topology (Figure 3.4) having more congestion. The star topology creates two TCP sessions, with each session over 2 hops. For the star topology, we measured the worst-case throughput over the paths, i.e., the throughput for the slowest session. For this experiment, broadcast ACKs were transmitted at the unicast rate. Figure 3.9 shows the throughput as a function of data rate. The X-axis shows data rate measured in Mbps, and the Y-axis shows the end-to-end throughput in Mbps.

As expected, BA shows more improvement in both scenarios. For the 3-hop linear topology, the maximum throughput gap between BA and UA is 12.2% as compared to 10% for 2-hop. As more relay nodes become involved in bi-directional aggregation, the overall aggregation ratio increases. For the star topology, the maximum throughput gap is 11% because network congestion leads to longer queues.

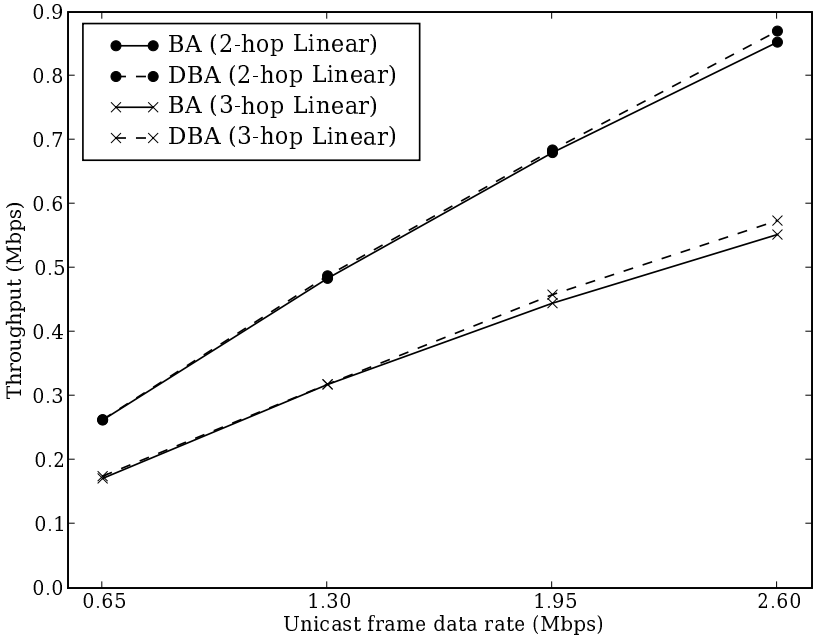


Figure 3.10: TCP for delayed BA

This allows BA to have more aggregation opportunities than UA because UA only supports aggregating frames being transmitted to the same receiver. Note, we also show the no aggregation result for 3-hop, which has the expected performance. In addition, we expect that, for star topologies, the result of no aggregation will be worse than that of UA and BA. This is clear because no aggregation does not support any methods that reduce the cost of TCP ACKs.

Delayed Aggregation

We designed this experiment to study the impact of the queueing delay on our aggregation approach. We expect that longer queues will result in more opportunities for aggregation.

We created a delayed version of BA, delayed BA (DBA), which forces relay

nodes to pause transmission until they have 3 frames in their queues. We choose 3 frames because that is the maximum number of TCP data frames that can be aggregated, given the parameters of our experiments. Figure 3.10 shows the performance comparison between BA and DBA over both of 2-hop and 3-hop linear topologies as a function of rate. The X-axis shows the unicast data rate in Mbps and the Y-axis shows the end-to-end throughput in Mbps.

We observed that the performance of BA and DBA is similar at the unicast rates of 0.65 Mbps and 1.3 Mbps and at higher rates DBA outperforms BA slightly. The maximum gap is 2% and 4% for 2-hop and 3-hop networks respectively. This improvement is smaller than we expected.

Forward vs. Backward Aggregation

We defined "forward aggregation" to mean the aggregation of packets going in the same direction, and "backward aggregation" to mean the aggregation of packets flowing in the opposite direction, such as TCP data and ACKs. We designed this experiment to gain some insight about where the benefits from backward aggregation occur. We do this by disabling forward aggregation so that the benefit from aggregation comes purely from combining TCP data and ACKs into one transmission.

For this experiment, the 3-hop linear topology was used. Figure 3.11 compares the performance of no aggregation, BA, and BA without forward aggregation. The X-axis shows the data rate in Mbps and the Y-axis shows the end-to-end throughput in Mbps.

The results show that the performance gap between BA and BA with disabled forward aggregation becomes bigger as the unicast data rate increases. This means that backward and forward aggregation affect the throughput equally when low data rates are used. As the data rate increases, forward aggregation has a greater

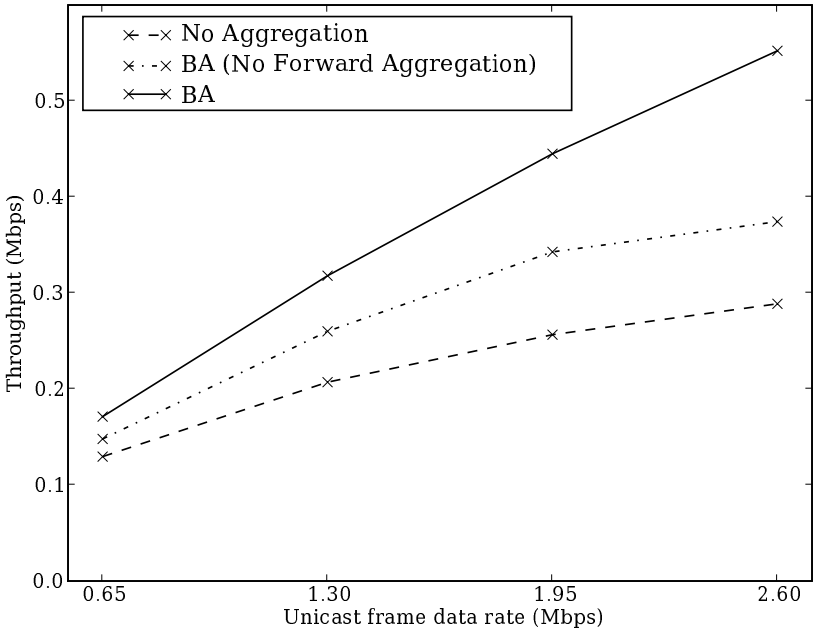


Figure 3.11: TCP without forward aggregation

impact on the throughput. This probably reflects a limit in the level of bi-directional aggregation possible in our benchmarks.

Detailed Analysis

Although the results shown thus far show the basic performance characteristics of our system, a more detailed analysis can give us more insight about the performance of DBA, BA, UA, and NA. We start by focusing on the impact of bi-directional aggregation in a 2-hop network, and then we extend our analysis to star and 3-hop networks to gain insight about the impact of the topology on performance.

2-hop Linear Topology Table 3.2 shows the average frame size (in bytes), the number of transmissions (TX) (as a percentage of the no aggregation number), and

Table 3.2: 2-hop relay node detail

	NA	UA	BA	DBA
Frame Size	765B	2662B	2727B	3477B
Total TXs	100%	33.7%	26.7%	21.1%
Size overhead	15.1%	6.83%	6.55%	5.8%

the size overhead (as a percentage of the MAC and PHY header size compared to total size) of NA, UA, BA and DBA in a 2-hop network. The table shows that the average frame size increases dramatically when we introduce aggregation and somewhat more modestly when broadcast and then delayed aggregation are introduced. A maximum TCP data frame is 1464B, and a TCP ACK is 160B, the average of which is 812B. This is close to the average NA size, suggesting that some TCP data segments are smaller than maximum size. On the other hand the average size for UA is 2662B which divided by 812B is 3.3. For the parameters we are using, 3 maximum size TCP data segments will fit in one frame, which suggests that for UA we are typically fully aggregating data frames and perhaps sometime sending as many as 4 ACKs at once. Thus TCP effectively expands its window to take advantage of aggregation. BA and DBA both achieve slightly better aggregation, as expected. The results for transmissions bear this analysis out since aggregating 3 data or 3 ACK frames in each transmission would result in one-third the number of transmissions, exactly as seen. Again, the greater aggregation of BA and DBA result in somewhat fewer transmissions compared to UA. Finally, the differences in header overheads reflect these same trends as we would expect.

Table 3.3 presents the average percentage time overhead as a function of the data rate. The overhead includes the transmission time for MAC and PHY headers, the transmission time for control frames, backoff, DCF interframe space (DIFS), and SIFS. This table shows very clearly how at higher data rates overhead begins to dominate. In the no aggregation case it goes from 22.4% to 52.1%. It also shows

Table 3.3: 2-hop relay node time overhead

Data Rate	NA	UA	BA	DBA
0.65	22.4%	6.7%	5.8%	5.2%
1.3	34.9%	14.3%	11.4%	10.3%
1.95	44.4%	19.3%	15.5%	14.3%
2.6	52.1%	24.8%	19.9%	17.7%

Table 3.4: Relay node frame size

	2-hop	Star
UA	2662B	2651B
BA	2727B	3432B

that aggregation is very effective in reducing this overhead.

Star Topology vs. 2-hop Linear Topology Table 3.4 shows the average frame size in both 2-hop and star topologies. The frame size is almost constant for UA, reflecting the fact that only frames with the same destination can be aggregated, and thus there are no greater possibilities for aggregation in the star topology than in the 2-hop one. For BA there is a substantial increase for the star because TCP ACKs destined for node 3 and 4 can be aggregated together and with TCP data frames destined for node 1. Table 3.5 shows the size overheads for the two topologies. It shows a similar trend as frame size. Table 3.6 shows the number of transmissions relative to the no aggregation case for the two topologies. It shows similar trends as the previous two results, although UA shows fewer transmissions for the star

Table 3.5: Relay node size overhead

	2-hop	Star
UA	6.83%	6.83%
BA	6.55%	5.93%

Table 3.6: Relay node transmission percentages

	2-hop	Star
UA	33.7%	30.7%
BA	26.7%	22.5%

Table 3.7: Frame size at all nodes for 2-hop and 3-hop networks

	Server (2^+)	Relay (2)	Client (2)	Server (3)	Relay1 (3)	Relay2 (3)	Client (3)
UA	3897	2662	463	3451	2384	2224	443
BA	3488	2727	447	3313	2538	2670	430

$^+$ indicates the number of hops.

topology. This is because UA suffers from queue overflow, which results in sending fewer packets for a given file size. On the other hand, for NA, we multiply the total number of transmissions by two because we do not have the NA results for the star topology. These decrease the transmission percentage for UA.

3-hop Linear Topology vs. 2-hop Linear Topology Table 3.7 shows the frame size of the TCP server, relay node(s), and TCP client in both 2-hop and 3-hop networks. For both BA and UA, the frame sizes at the TCP server and client imply that the server transmits an aggregated frame containing two (2928B) or three (4392B) subframes and, as a response, the client sends two (320B) or three (480B) ACKs back to the server.

The data shows that, at the relay nodes, the average frame size in a 2-hop network is bigger than that in a 3-hop network. This is because the TCP data and TCP ACK transmission rates decrease for 3-hops. The reduced transmission rates affect the average sizes at the TCP server and client. The average frame sizes of BA at the TCP server and client are 3488B and 447B in a 2-hop network respectively. Meanwhile, the average frame size decreases to 3313B and 430B in a 3-hop network. These reduced sizes make the average frame size at the relay nodes

decrease. However, it is useful to compare the sizes at the relay nodes. For 2 hops, the difference between UA and BA is 65B, while for 3 hops at relay1 it is 154B and at relay2 it is 446B. Thus we see a significant increase in the amount of aggregation as the number of hops increases.

Chapter 4

Rate-Adaptive Aggregation

In the previous Chapter, we developed a design that improves efficiency by taking advantage of frame aggregation and the broadcast nature of wireless transmissions. From the experimental results, we observed that this design improves throughput by amortizing overheads over several frames and becomes more effective as data rate increases. This implies that rate adaptation has an important role to play in the performance of frame aggregation. Thus, to improve network performance, it is important to maximize synergy between rate adaptation and frame aggregation.

Here, we present a new rate-adaptive frame aggregation scheme that combines both rate adaptation and frame aggregation. We start by presenting general related work. Then we present experimental results that inform our design and the details of our design for adapting the aggregation size with rate adaptation. Finally we present a performance evaluation of our approach.

4.1 Related Work

There have been several cross-layer approaches that adjust the allowable aggregation size and channel rate jointly [12, 35, 36]. Sadeghi *et al.* [12] proposed opportunistic

autorate (OAR), the details of which have been discussed in Section 2.2.4. Kim *et al.* [35] proposed a rate-adaptive protocol with dynamic fragmentation, which increases the fragment threshold for higher rates, and thus allows a wireless system to send more data at higher rates. Different from OAR, at favorable channel conditions, this saves header overheads by increasing the fragment threshold instead of sending multiple packets. Chen *et al.* [36] suggested rate-adaptive framing (RAF), which controls channel rate and frame size jointly for wireless networks. Based on interference patterns, the RAF receiver determines the optimal channel rate and frame size by computing the throughput for all possible rates and sizes and then piggybacks this rate and size on the ACK. The RAF transmitter applies this information to the next data transmission.

Our design differs from these approaches in that our scheme allows for adapting the rate and aggregation size without significant computation overhead, and it does not require calibration for adaptation.

4.2 Pre-Design Experiments and Analysis

The goal of the analysis and experiments in this Section is to gain insight into what our design should be. Two main questions arise: how to bound aggregation size and whether it is important to acknowledge individual frames in an aggregate.

Aggregating multiple frames into a single transmission improves throughput by amortizing overhead over more data. However, as the aggregation size grows, the additional improvement becomes limited. Further, if there are multiple TCP sessions in a wireless network, more aggregation of a session might increase round-trip time (RTT) variation of other sessions, which might decrease TCP transmission rates [37]. Thus, bounding the aggregation size is important to improve overall network performance. Here, one question is how we should bound the aggregation size. We examine this by studying the impact of frame aggregation on overhead.

A second question is how different channel conditions effect the performance of frame aggregation. In particular, for a time varying channel, the aggregation size that maximizes throughput could depend on how long it takes the channel to become uncorrelated. We examine this question by measuring the throughput as a function of aggregation size for a variety of node speeds.

A third question is how block ACKs, which allow individual frames to be acknowledged, effect rate adaptation and frame aggregation. Block ACKs require the additional cost for bitmaps but have an advantage when partial drops happen. Thus, we examine the impact of block ACKs on rate adaptation and frame aggregation for a variety of channels. We begin by studying the impact of block ACKs on rate adaptation when frame aggregation is used. Then, for frequency selective channels, we study the impact of block ACKs on frame aggregation by measuring throughput as a function of aggregation size for the schemes using block ACKs and not using block ACKs.

4.2.1 Size Adaptation

We designed this analysis and experiments to study the impact of aggregation size on performance for time-invariant frequency flat and time varying channels. Here, we excluded experiments for frequency selective channels because we found that the best aggregation size is insensitive to those channel effects due to block ACKs, as we will see in Section 4.2.2.

Time-invariant Frequency Flat Channel

We begin by calculating the impact of frame aggregation on overhead for a simple time-invariant frequency flat channel for a variety of rates and for both single and double streams of data. Overhead is the percentage of total transmission time spent on overhead rather than sending actual data. For this analysis, we used a time-

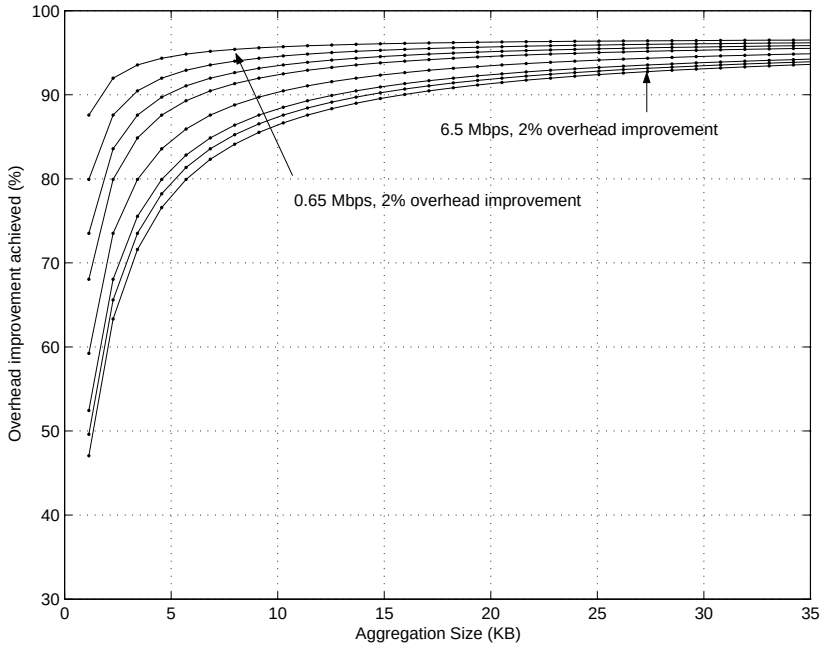


Figure 4.1: Overhead analysis for single stream rates

invariant frequency flat channel to minimize the impact of the channel on overhead.

Figure 4.1 shows the inverse of the overhead (that is the percentage of useful data) as a function of aggregation size for all possible single stream rates. The X-axis is the aggregation size in KB, and the Y-axis is the percentage of useful data. In each graph, the lowest data rate is at the top, the highest at the bottom, and the other rates are in-between. Arrows indicate where the overhead improvement reaches 2%. Figure 4.2 displays the same information for the MIMO double data stream case.

At all rates, when there is no aggregation, overhead is significant. However, as the aggregation size grows, the improvements rapidly reach a point of diminishing returns, as shown by the 2% overhead indications for the lowest and highest rate. In addition, as expected, the benefits of frame aggregation are greater for higher rates, and for higher rates the point of diminishing returns is not reached until significantly

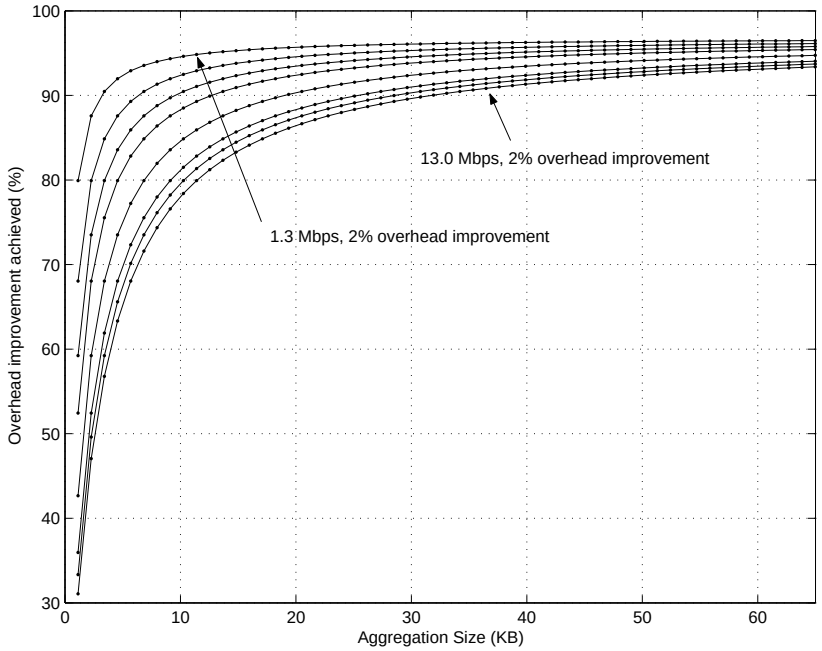


Figure 4.2: Overhead analysis for double stream rates

greater amounts of data are aggregated. The double stream graph shows that even greater improvements are possible in this case.

These results suggest that a frame aggregation system should bound the aggregation size since allowing arbitrary sizes will result in limited improvements. However, exactly what bound to use should vary with rate, reflecting the need to aggregate larger amounts of data to attain a particular throughput gain for higher rates. For our design, we use a 2% overhead bound for each data rate as an initial maximum aggregation size.

Time Varying Channels

This experiment examines the impact of the channel coherence time on the best aggregation size for time varying channels. We studied this by measuring throughput as a function of aggregation size for the single stream rates of 2.6 and 5.85 Mbps,

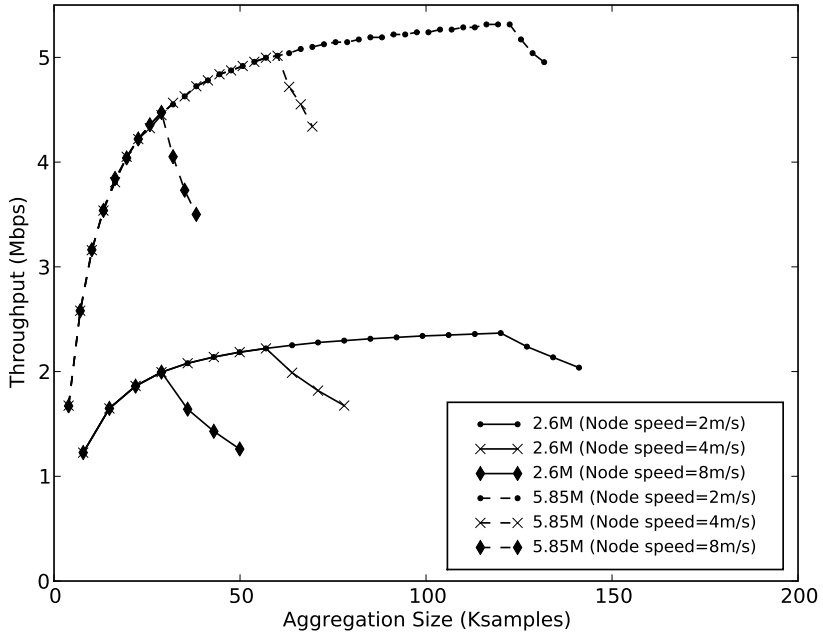


Figure 4.3: Throughput vs. Aggregation size (in physical samples) as a function of node speed

and for the node speeds of 2, 4, and 8 m/s. As discussed above, we used a channel emulator to create the time varying channels.

Figure 4.3 shows the throughput as a function of aggregation size in Ksamples. Samples are what the PHY actually uses when transmitting data. When a higher rate modulation is used, more bits fit into a sample. The X-axis is the aggregation size in Ksamples and the Y-axis is the throughput in Mbps. In addition, Figure 4.4 shows the throughput as a function of aggregation size in bytes. The X-axis is the aggregation size in Kbytes and the Y-axis is the throughput in Mbps. We performed each set of experiments 5 times and found that the standard deviation was negligible (i.e., order of 10^{-2} or 10^{-3}).

The results show that, for a given node speed, throughput starts falling off

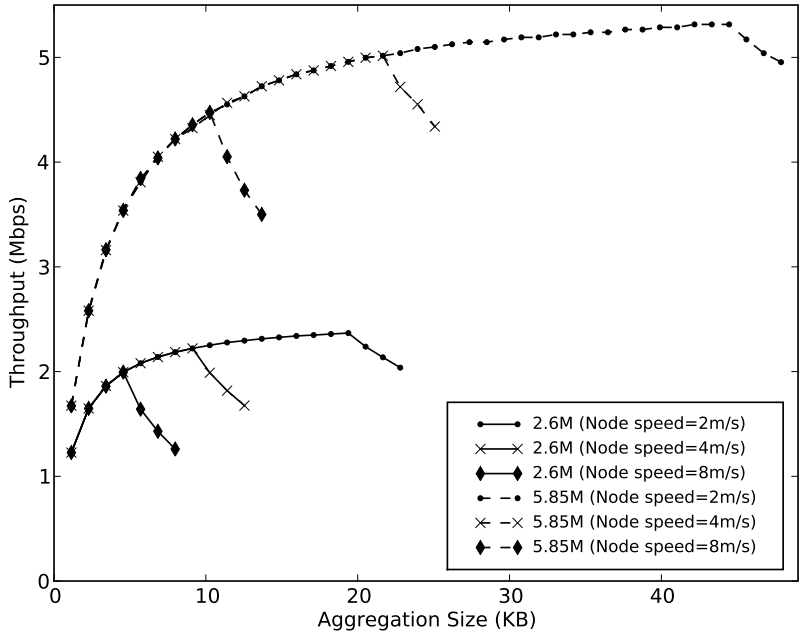


Figure 4.4: Throughput vs. Aggregation size (in bytes) as a function of node speed

at the same number of samples regardless of data rate. This means that the best aggregation size (in physical samples) corresponds to when the channel begins to be uncorrelated. Thus, it is possible to transmit this many samples before the channel changes. Further, the faster the channel changes, the fewer samples can be sent before the channel is uncorrelated. When we change the X-axis to the byte domain, the result shows that, for a given coherence time, the best aggregation size (in bytes) increases as data rate increases. This is because using higher rates allows each sample to carry more information per sample.

This result suggests that adapting the aggregation size for time varying channels is required to maximize throughput. The best aggregation size in physical samples corresponds to the channel coherence time, and, further, the best size in bytes varies with data rate. This gives us the insight that size adaptation needs to detect

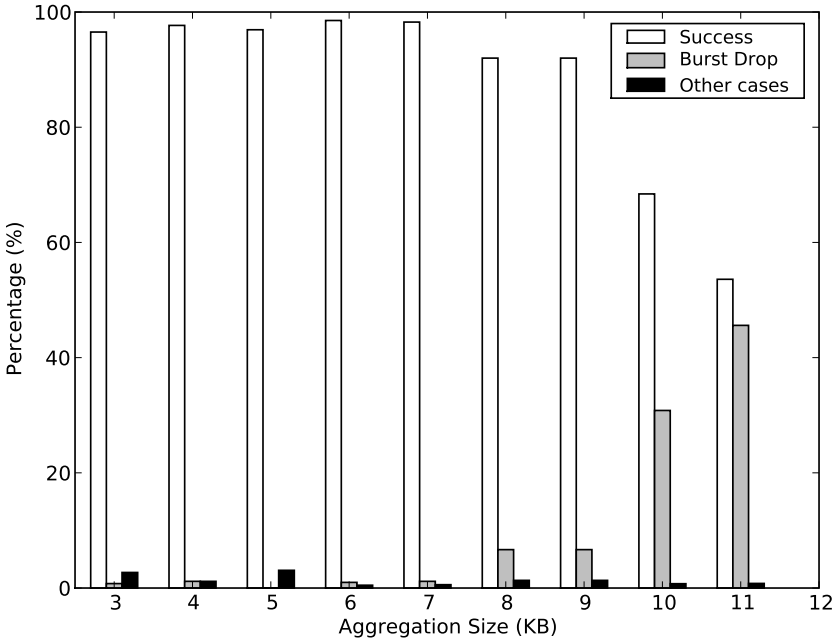


Figure 4.5: Failure pattern for the single stream rate of 1.95 Mbps

the channel coherence time in samples and further, for each rate, the best size in bytes can be obtained by converting the best size in samples into the byte size based on the number of bits that a single sample carries.

The dependence on the channel coherence time means our design needs some way to discover what the coherence time is. We address this by analyzing the packet failure pattern caused by using an excessive aggregation size.

We performed a series of experiments, increasing the aggregation size by 1KB. For our experiments, we created a real time-varying channel, the details of which are described in Section 2.4.1. To minimize packet errors caused by channel noise and collisions, we used a high average SNR ($\sim$ 28 dB) to support the rate, and we used only two nodes, a transmitter and a receiver. Figure 4.5 shows the percentage of the total number of success, burst drops, and other cases as a function of the

aggregation size for the rate of 1.95 Mbps. Burst drops are a burst of successful subframes followed by a burst of erroneous subframes. The failure pattern for other rates was similar to this and so has not been presented. The X-axis is the aggregation size in Kbytes, and the Y-axis is the percentage of each case.

The result shows that burst drops increase as aggregation size grows. This means that excessive aggregation mostly results in burst drops. This is because once channel variation causes a subframe to be dropped, all the other subframes following the first one will also be impossible to decode.

This result suggests that we can track the channel coherence time by using the burst drop as an indicator of excessive aggregation size. In particular we can use the location of the first dropped subframe as a measure of how many symbols we can transmit correctly.

4.2.2 Block ACKs

Block ACKs carry a bitmap to acknowledge individual subframes. We designed these experiments to study the impact of block ACKs on rate adaptation and frame aggregation for time-invariant frequency flat and frequency selective channels. We found that, once the correct aggregation size was found, using block ACKs does not have an effect on time-varying channels.

Time-invariant Frequency Flat Channel

This experiment examines the impact of block ACKs on rate adaptation for a time-invariant frequency flat channel. We did this by measuring the rate transition points used for rate adaptation as a function of aggregation size for the aggregation schemes using block ACKs and not using block ACKs.

We performed Monte-Carlo simulations using a AWGN channel model, as described in [38]. We assumed that each subframe's error is uncorrelated and that

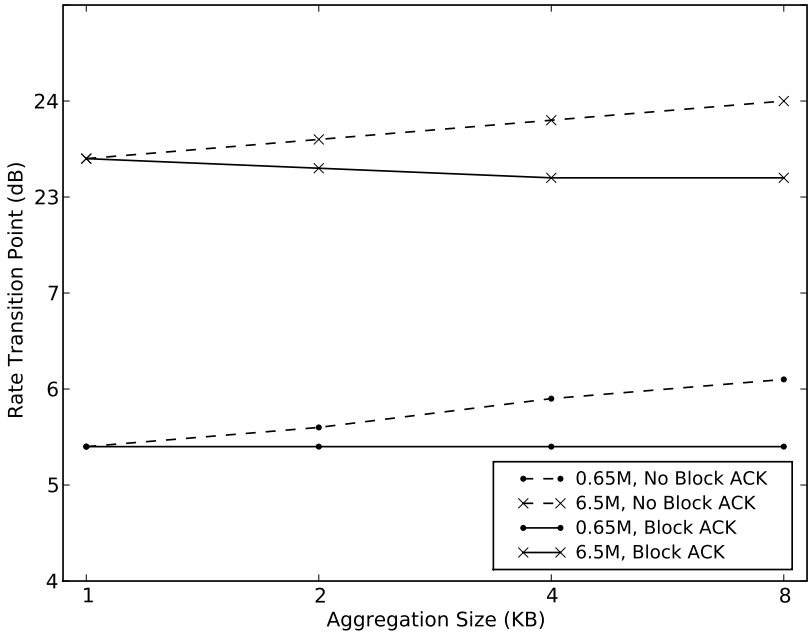


Figure 4.6: Rate transition points (in dB) for rate adaptation

the size of each subframe is fixed at 1KB. We chose the rate transition points by measuring throughput using the fixed rates as a function of SNR for a variety of aggregation sizes and for the aggregation schemes using block ACKs and not using block ACKs. Figure 4.6 shows the lowest and highest rate transition points (in dB) as a function of aggregation size for the schemes using block ACKs and not using block ACKs. The X-axis is the aggregation size in Kbytes, and the Y-axis is the rate transition point in dB.

The result shows that, for the aggregation size of 1KB (no aggregation), the schemes using block ACKs and not using block ACKs have the same rate transition points because partial drops do not happen. As the aggregation size increases by a factor of 2, the scheme not using block ACKs shifts the lowest and highest rate transition points toward higher SNR by 0.2 dB. This is because, for a given rate,

low SNR increases partial drops, which causes entire packets to be dropped. Thus, as SNR decreases, the throughput falls off rapidly, resulting in a shift of the rate transition points toward higher SNR. On the other hand, for the scheme using block ACKs, aggregation does not change the lowest transition points, while it shifts the highest transition point toward lower SNR slightly. This is because using block ACKs makes aggregation robust on partial drops, which prevents the throughput from being degraded rapidly as SNR decreases. In addition, in high SNR regions, aggregation increases the throughput and becomes more effective for higher rates.

This result suggests that using block ACKs allows rate adaptation and frame aggregation to be decoupled. This is because, using block ACKs, we can use the rate transition points for non-aggregated frames, which in the worst case will be slightly conservative for some rates and larger aggregates.

Frequency Selective Channels

This experiment examines the impact of a block ACK scheme on frame aggregation for frequency selective channels. We did this by measuring throughput as a function of aggregation size for the schemes using block ACKs and not using block ACKs.

For our experiment, we created frequency selective channels using our channel emulator. The average SNR of the channel is 30 dB. We used root-mean-square (RMS) delays of 15 and 150 ns, which are chosen based on the IEEE 802.11n channel model [39]. The 15 ns RMS delay represents a small office environment, and the 150 ns RMS delay represents a large open space. The number of channel taps is set to 18, which follows IEEE 802.11n channel model, and each tap duration is set to a single sample duration. Figure 4.7 shows the throughput as a function of aggregation size for the schemes using block ACKs and not using block ACKs. The X-axis is the aggregation size in Kbytes, and the Y-axis is the throughput in Mbps. For our experiments, we performed each set of experiment 5 times, and then put

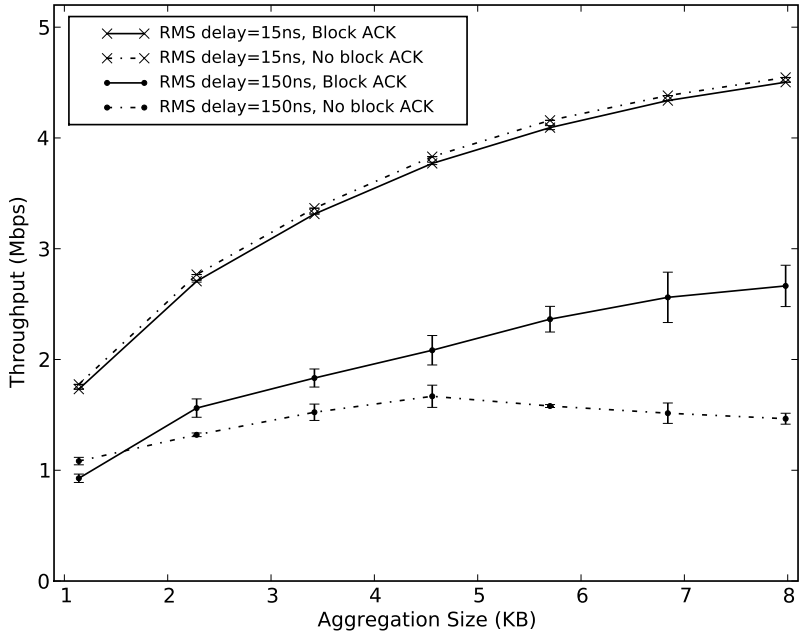


Figure 4.7: Throughput vs. Aggregation size on a frequency selective channel (Data rate=6.5 Mbps)

error bars on the graphs.

The result shows that, for the RMS delay of 15 ns, the aggregation scheme not using block ACKs achieves slightly better throughput than the scheme using block ACKs. This is because, for the given SNR, partial drops rarely happen on this frequency-flat channel. Block ACKs include the additional overhead of the bitmap which is only a benefit if partial drops occur. For the RMS delay of 150 ns, the scheme using block ACKs increases throughput monotonically as the aggregation size grows. On the other hand, for the scheme not using block ACKs, the throughput increases initially but, at some point, it starts falling off. This is because, for the scheme not using block ACKs, partial drops cause the entire packet to drop and the probability of a partial drop increases as the aggregation size increases.

This result suggests that using block ACKs facilitates the use of large maximum frame aggregation sizes in frequency selective channels. Further, if block ACKs are used, there is no need to adapt the maximum aggregation size to support frequency selective channels.

4.3 Design

There are a variety of ways to combine rate adaptation and frame aggregation. In theory, it is desirable to find the joint optimal rate and size. One possible way to do this is to find the optimal point by searching all possible rates and sizes. However, this might cause significant computation overhead as the number of possible rates and sizes increase.

Our pre-design experiments suggested that, if we use block ACKs, we can decouple rate adaptation from frame aggregation without a significant penalty. This means that, for higher rates, the rate cutoffs are a little conservative as the aggregation size grows. However, using block ACKs allows us to reduce the design cost by decoupling rate adaptation from frame aggregation. Thus, our design uses block ACKs and provides a way to adapt the aggregation size on top of rate adaptation. Our strategy is that a rate adaptation algorithm determines a data rate based on the channel and then, given a rate, the MAC adapts the aggregation size based on the received data frames.

For our design, any kind of rate adaptation technique could be used as the underpinning of our size adaptation. Here, the specific rate adaptation algorithm we used is RBAR [22], but it could be replaced by other approaches such as SoftRate [40] or robust rate adaptation algorithm (RRAA) [41] without affecting our results in any important way. For RBAR, we choose the rate transition points by experimental calibration using the fixed rates supported by Hydra as a function of SNR. Fixing the rate transition points, the MAC first chooses the data rate based on the received

SNR from a RTS. Then, the MAC obtains the aggregation size for that rate from the adaptation control module and piggybacks this information on the CTS. The details of the implementation of our design have been described in Appendix B.

4.3.1 Automatic Size Adaptation

We have seen from the pre-design experiments that the best aggregation size varies with the channel coherence time. The remaining design question is how to dynamically determine what the best size is. Since our pre-design experiments show that the correct size in samples does not vary significantly with rate, our design tracks the correct size in samples. Once the rate has been determined, the number of bits per sample at that rate is used to convert the optimal size in samples so that the size in bytes can be returned on the CTS.

Our pre-design experiments give us clear guidance as to how to determine the correct size in samples by examining the incoming packets for burst drops. If a burst drop occurs then we can use the location where it starts as an indication of when the channel has diverged. Since the channel varies with different transmitters, each receiver keeps a table of the best aggregation size for each transmitter. If a burst drop occurs, then the table is updated.

If no transmissions have been received from a transmitter we must know how to initialize the table. We do this by using the 2% overhead bound as the initial maximum size. It the channel is not time-varying, no burst drops will occur and this will remain the bound. Otherwise, the initial transmission will result in a burst drop and the correct value will be stored in the table.

Another issue is that the coherence time might increase or decrease. If it decreases, a burst drop will occur and the table will be updated correspondingly. To account for possible increases, we periodically increase the size to probe for a new value, much as in TCP or ARF. The protocol counts the number of transmissions

without a burst drop and when a threshold is reached, it increments the maximum aggregation size. One important detail is that it avoids incrementing the count when the transmission is smaller than the current maximum aggregation size.

Now, we consider how size adaptation works in some exceptional cases. A possible case is when the channel statistics suddenly change. If the channel becomes fast-changing, then a burst drop will occur and thus our size adaptation adjusts to the best size immediately after the channel change. On the other hand, if the channel becomes slow-changing, then our size adaptation increases the best size by probing the channel, and eventually reaches the best size for the slow-changing channel. Our scheme allows fast adaptation to the best size by detecting burst drops, which makes our size adaptation differ from TCP or ARF.

Another possible case is when there is interference from hidden nodes. We are using block ACKs and thus interference causes drops for erroneous packets only. In addition, in some cases, the partial drops might have the same pattern as a burst drop, resulting in incorrect estimation of the best size. However, this is not a problem because our size adaptation will eventually return to the correct size.

4.3.2 Design Issues

Some additional design issues flow from the basic design described above.

Queue Management

The standard DCF MAC sends the data at the head of queue. If it is lost, the MAC retransmits this data until the maximum retry count is reached. For retransmission, the contention window, used for random backoffs, increases exponentially based on the retry count. However, for our rate-adaptive frame aggregation scheme, the aggregation size might change for the retransmission, which allows the MAC to aggregate more or fewer subframes for the retransmission. Thus, to utilize this

feature, we modified the MAC by the following design principle: first that all transmissions use the same semantics, and second that frame aggregation should be an optimization and not fundamentally alter the basic semantics of transmission. That is, for both the transmission and retransmission, the MAC searches its queue, assembles subframes based on the aggregation size, and then sends the assembled subframes in a single transmission. The contention window used for random backoff is determined by the retry count of the data at the head of the queue. Further, failed transmissions only increase the retry count of the data at the head of the queue.

Virtual Carrier Sensing

Virtual carrier sensing is a technique to reserve the floor by announcing the duration of a data transmission. For this purpose, the IEEE 802.11 MAC uses a RTS/CTS exchange [1] which contains the duration for the following data transmission. Based on this exchange, the neighbors within the transmission ranges of a sender and receiver defer their own transmissions.

Our design also uses the RTS/CTS exchange for virtual carrier sensing, but it requires modest changes from the standard. This is because the rate and size for a data transmission are determined by the receiver, and thus the RTS cannot contain the exact duration of the data transmission. Thus we introduce receiver-initiated virtual carrier sensing. This scheme initiates the network allocation vector (NAV) update on the receiver side. When receiving the RTS, the receiver calculates the duration for the CTS based on the rate and size chosen for the following data transmission. This allows the neighbors overhearing the CTS to update their own NAV based on the correct duration. In addition, the neighbors within the transmission range of the transmitter update their NAV by overhearing the data transmission, which also carries the correct duration.

4.4 Experimental Results

We performed a series of experiments using a variety of channels to validate our technique. The details of experimental setup have been presented in Section 2.4. Here, the MAC frame size increases by 4 bytes because the frame format used for our design follows the IEEE 802.11n standard which has a 4-byte prefix field. Thus, a 1KB UDP packet results in 1140B MAC frame and a 1357-byte TCP packet results in a 1468B MAC frame. The details of MAC frame format have been presented in Appendix B. In this Section, we present performance results on both real and emulator-based channels.

4.4.1 Time-invariant Frequency Flat Channels

This experiment examines the impact of our technique on throughput for a real time-invariant frequency flat channel. We did this by comparing the throughput for the rate adaptation schemes using no aggregation, using a fixed aggregation size of 8KB for all data rates, and using an overhead bound for each data rate. The overhead bound is set to the aggregation size achieving 2% overhead improvement for each data rate. For the scheme using a fixed aggregation size, we chose 8KB, which is the overhead bound for the lowest rate of 0.65 Mbps. Figure 4.8 shows the throughput as a function of SNR. The X-axis is the average SNR in dB and the Y-axis is the throughput in Mbps. The gray lines show throughput for fixed data rates.

The result shows that the aggregation scheme using a fixed aggregation size of 8KB achieves significant improvement over no aggregation for high SNR where a high rate is used. This is because overhead becomes more important for higher rates, which makes frame aggregation more effective. Further, the aggregation scheme using the overhead bound achieves more performance gain over the scheme using a fixed aggregation size for higher rates by allowing more aggregation for those rates.

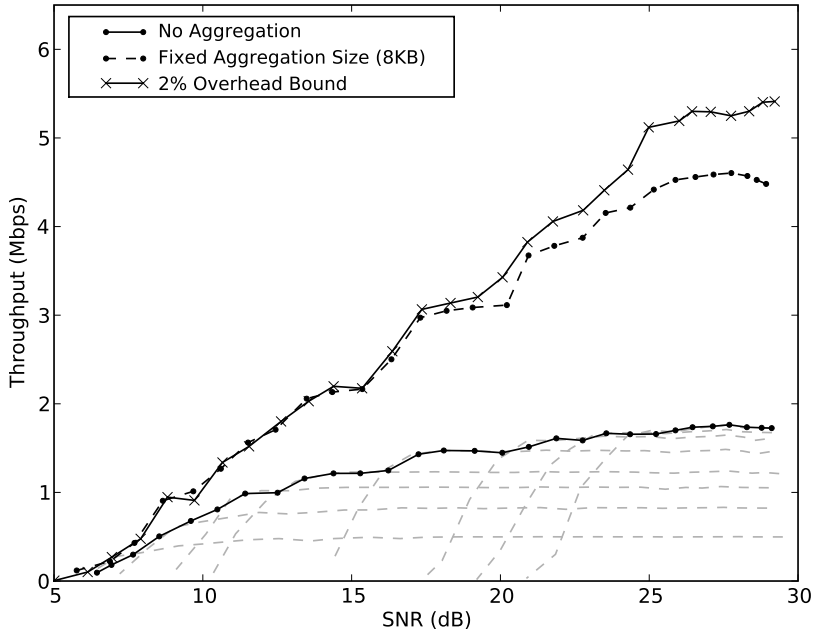


Figure 4.8: Throughput as a function of SNR for a time-invariant frequency flat channel

On the other hand, the performance gap becomes small in low SNR regions. This is because, in these regions, rate adaptation chooses low rates, where the impact of overhead becomes negligible.

4.4.2 Frequency Selective Channels

Two important issues are how block ACKs influence our technique and how frequency selective channels influence our technique. This experiment examines these issues by comparing the throughput between our aggregation schemes using block ACKs and not using block ACKs. For our experiment, we created a frequency selective channel having an RMS delay of 150 ns by using our channel emulator.

Figure 4.9 shows the throughput as a function of SNR. The X-axis is the

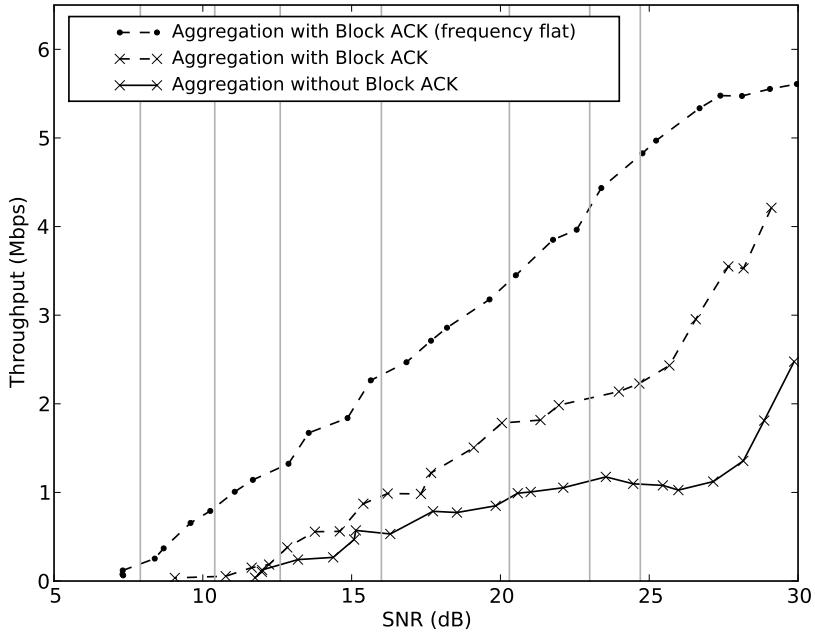


Figure 4.9: Throughput as a function of SNR for a frequency selective channel

average SNR in dB, and the Y-axis is the throughput in Mbps. The vertical gray lines show the rate transition points. In addition, we have included the experimental result for our aggregation scheme for a frequency flat channel as a point of comparison.

The result shows that our aggregation scheme using block ACKs outperforms the scheme not using block ACKs in all SNR regions. This is because our scheme using size adaptation raises the aggregation size (in bytes) when the channel is good. A larger aggregation size is vulnerable to channel errors, and partial ACKs mitigate that vulnerability. In addition, for SNRs above 25 dB, the throughput of both schemes decreases exponentially as SNR decreases, while below 25 dB, the degradation slows. This is because, above 25 dB, most data transmissions use the highest rate of 6.5 Mbps and thus the throughput is only a function of SNR. On

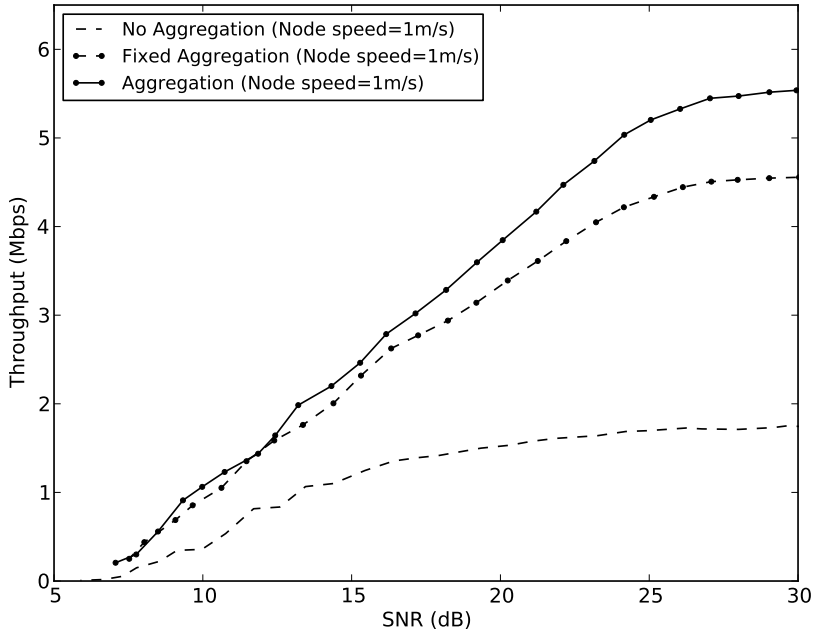


Figure 4.10: Throughput as a function of SNR for a slow time varying channel

the other hand, below 25 dB, rate adaptation reduces the degradation by choosing a robust rate based on the channel condition.

4.4.3 Time Varying Channels

These experiments examine the impact of our technique on throughput for time-varying channels. We did this by comparing the throughput for the rate adaptation schemes using no aggregation, using a fixed aggregation size of 8KB for all rates, and using our size adaptation. We used a channel emulator to create a variety of time-varying channels.

We are using RBAR, an explicit feedback scheme, which requires that the channel coherence time be longer than the time between the RTS and data transmission. This channel coherence time corresponds to a maximum node speed of

8.27 m/s. For this calculation, we assume that a 1140B data frame is transmitted at the base rate of 0.65 Mbps. Thus, for our experiments, we used the node speeds of 1 m/s, 4 m/s, and 8 m/s which are below this threshold. 1 m/s represents walking speed, 4 m/s represents running speed, and 8 m/s is a slow car speed. Figure 4.10 shows the throughput as a function of SNR for a slow time-varying channel with the node speed of 1 m/s. The X-axis is the average SNR in dB, and the Y-axis is the throughput in Mbps.

The result shows that our technique achieves better throughput than the schemes using a fixed aggregation size of 8KB in high SNR regions. Further, we observe that performance gain becomes significant as the average SNR increases. This is because, for a higher SNR, rate adaptation chooses a higher rate, which allows more aggregation for a given channel coherence time. However, below 11.7 dB, both the schemes achieves similar performance. This is because, for a lower SNR, rate adaptation chooses a lower rate, which allows less aggregation for a given channel coherence time, and thus 8KB is close to the best aggregation size.

Now, we consider how our technique works for faster time-varying channels. Figure 4.11 shows the throughput as a function of SNR for no aggregation and our technique for time-varying channels with the node speeds of 4 m/s and 8 m/s. We have also included the result for 4-m/s using a fixed aggregation size of 8KB. The X-axis is the average SNR in dB, and the Y-axis is the throughput in Mbps.

The result shows that our technique achieves better throughput than the schemes using no aggregation and using a fixed aggregation size in all SNR regions. Comparing our technique with the no aggregation scheme, we observe that the performance gap increases as the average SNR increases. This is because, for a higher SNR, rate adaptation chooses a higher rate, which allows more aggregation for a given channel coherence time. Further, comparing our technique with the scheme using a fixed aggregation size, at an SNR of 13.3 dB, both the schemes achieves sim-

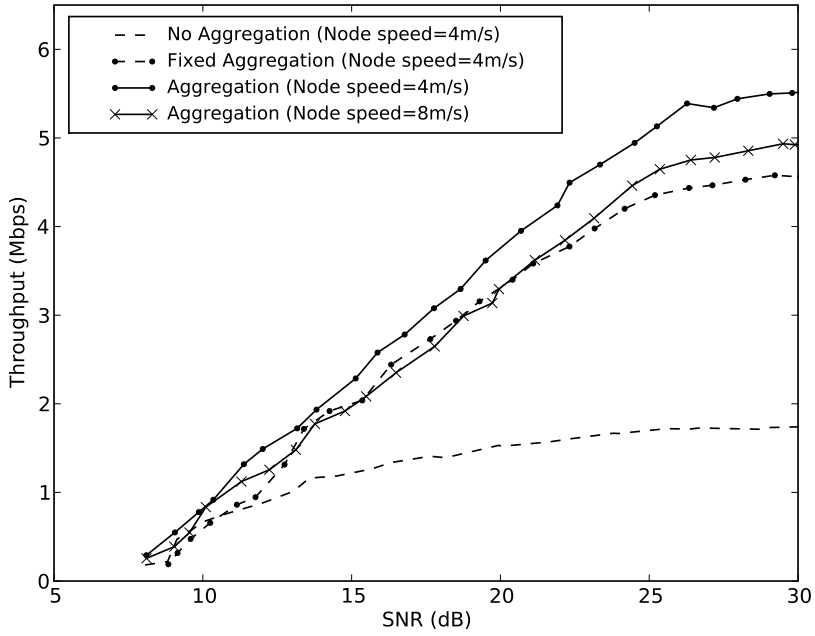


Figure 4.11: Throughput as a function of SNR for time varying channels

ilar performance. This is because 8KB is close to the best aggregation size for the given SNR and channel coherence time. However, above 13.3 dB, the performance gap increases as the average SNR increases. This is because our scheme allows more aggregation for higher rates. In addition, below 13.3 dB, the performance gap increases as the average SNR decreases. This is because 8KB becomes excessive as the average SNR decreases, which results in degrading throughput by causing significant retransmissions. Further, below 10.2 dB, the fixed scheme using 8KB shows the worst performance. For the node speed of 8 m/s, the performance gap between no aggregation and our technique decreases. This is because our technique decreases the aggregation size by detecting that the channel coherence time is reduced.

In the previous graphs, it was hard to see the relative throughput improvement between our technique and the scheme using a fixed aggregation size of 8KB.

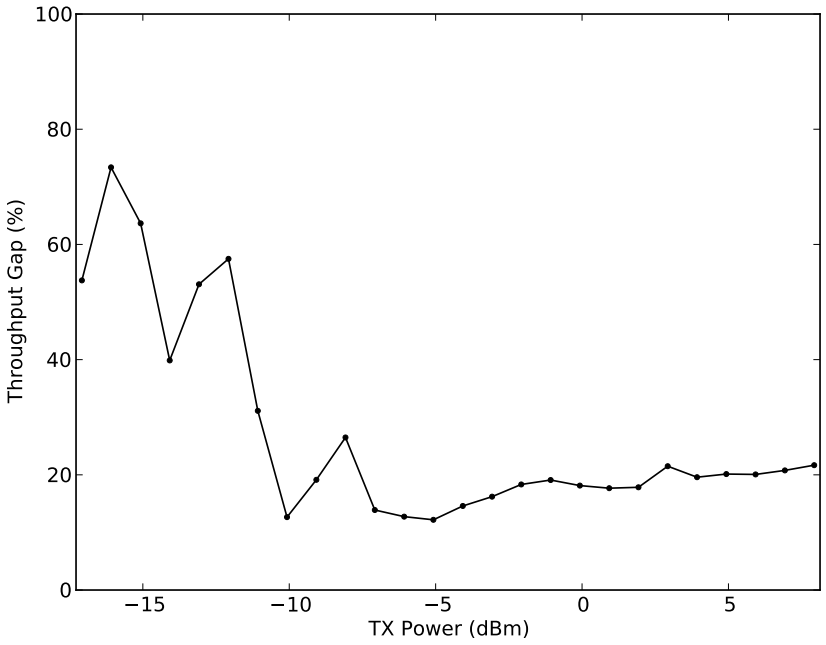


Figure 4.12: Throughput gap as a function of TX power for a time varying channel

To show the relative improvement, we calculated the throughput gap (as a percentage) between our technique and the scheme using a fixed aggregation size of 8KB based on the same experiment as for the throughput measurement. Figure 4.12 shows the throughput gap between our technique and the scheme using a fixed aggregation size of 8KB as a function of TX power for a time-varying channel with the node speed of 4 m/s. The X-axis is the TX power in dBm, and the Y-axis is the throughput gap in percentage. In the X-axis, the highest TX power corresponds to the SNR of 31 dB and the lowest TX power to 8.8 dB.

The result shows that our technique achieves more performance gain over the scheme using a fixed aggregation size of 8KB in low TX power (or SNR) regions. This implies that throughput loss caused by excessive aggregation size becomes more important than that caused by small aggregation size. The maximum throughput

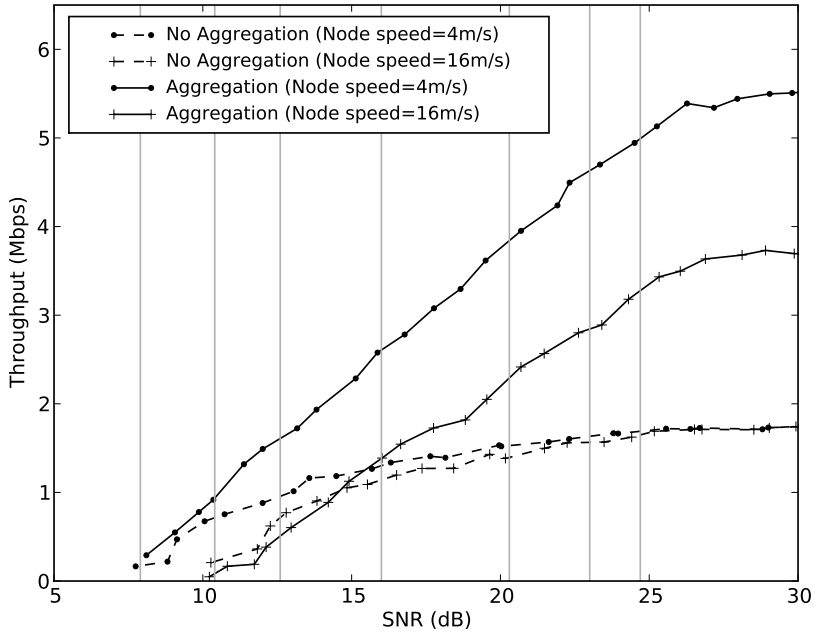


Figure 4.13: Throughput as a function of SNR for a fast time-varying channel

gap increases to 73.5 % when the TX power (or the average SNR) is -16.1 dBm (or 9 dB). In addition, we observe that the throughput gap fluctuates in low TX power regions. This is because the same TX power does not guarantee the same average SNR for each set of experiments.

Fast time-varying channel We were interested in seeing what the results would be if we included a channel that varied so fast that the assumption that it was fixed between the RTS and data transmission was violated. Figure 4.13 shows the throughput as a function of SNR for a fast time-varying channel. The X-axis is the average SNR in dB, and the Y-axis is the throughput in Mbps. For this experiment, we set the node speed to 16 m/s to create the fast time-varying channel. The vertical gray lines show the rate transition points.

The result shows that, for the no aggregation scheme, there is a performance gap between the node speeds of 4 m/s and 16 m/s. This is because the channel is changing very fast and thus rate adaptation does not work well. For our technique, the performance gap between 4 m/s and 16 m/s comes from two factors: first that rate adaptation does not work well for that channel, and second that the channel coherence time is reduced for the fast channel, which only allows small aggregation sizes. In addition, the result shows that, for SNRs between 10.4 dB and 15 dB, no aggregation outperforms our technique. This is because, in these SNR regions, the channel coherence time allows only a single 1140B frame, and our technique requires probing the channel to track the best aggregation size. For SNRs below 10.4 dB, the channel coherence time is not enough to transmit even a single 1140B frame. Thus, neither scheme works. In this case, we could consider using fragmentation, or we could add cyclic redundancy check's (CRCs) inside of individual subframes to allow retransmission of only portions of a subframe.

For our emulator-based experiments on time-varying channels, we use the Hydra platform which uses a 2 MHz bandwidth, while real 802.11 systems use 20 MHz. However, based on our system, we scaled the time variation for the channel and all the overheads to match a real 802.11 system. Thus, the performance gains we present are the same as they would be for 802.11. The only difference is that we present our throughput based on 2 MHz.

Chapter 5

Multi-Destination Aggregation

In this Chapter, we present an enhancement to our system to support multi-destination aggregation, where multiple unicast frames having different destinations and broadcast frames are aggregated into a single transmission. This extends rate-adaptive frame aggregation to broadcast and multiple unicast destinations. Thus, we can improve aggregation efficiency by aggregating more frames. This is important because multi-destination aggregation allows us to make better use of the space available in an aggregate.

We already have evidence that this is a good idea. The design presented in Chapter 3 allows a special case of multi-destination aggregation by treating TCP ACKs as broadcasts. From those experimental results, we observed that, for TCP streams, this design improves throughput by both amortizing header overhead and more importantly reducing the number of transmissions.

When does multi-destination aggregation become effective? A possible scenario is when there is an AP (access point) connecting to multiple devices. In this case, an AP might have queueing for packets going toward multiple devices, and thus multi-destination aggregation reduces overhead and the number of transmissions by aggregating them into a single transmission. Another possible (likely) scenario is

when a TCP application is used. The key observation is that TCP ACKs are small packets that flow in the opposite direction of the larger TCP data packets. We have already seen that our design presented in Chapter 3 allows multi-destination aggregation for TCP application and improves network performance by aggregating TCP data and TCP ACK traveling in the opposite direction.

Here our goal is to implement multi-destination aggregation in a general way. This means that we cannot ignore sending link-level ACKs for unicast frames as we did for TCP ACKs in Chapter 3. One question to be answered is whether this general design can achieve the same advantages that the special-purpose design had for TCP ACKs.

5.1 Design

Supporting multi-destination aggregation gives rise to a number of design issues. The goal of this design is to support multi-destination aggregation in general and to maintain the semantics of transmitting frames.

The first issue is that, to achieve our goal, the unicast packets in an aggregated frame will have to be acknowledged even if they have different destinations. However, typically link-level ACKs are transmitted immediately after packet reception. If a frame contains multiple destinations, the result of this strategy would be that multiple receivers would send link-level ACKs immediately after receiving the packet. These ACKs would then collide, resulting in retransmissions and thus increasing overhead.

The obvious solution to this problem is that not all the link-level ACKs can be sent at the same time. Thus, some ACKs will have to be delayed somehow. One possible solution is to use a centralized node to control the ACK transmission from multiple receivers in a scheduled manner. This would work in infrastructure-based networks. However, our goal is to control the ACK transmissions in a distributed

manner, which will allow our design to work even when there is no central controlling node.

The second issue is that, when we send multiple unicast frames having different destinations, there exist multiple channels between the transmitter and multiple receivers. Each of these channels might have a different channel quality. For the unicast frame at the head of queue, a sender transmits a probe to learn the data rate for the destination of the head of queue. However, this seems not to be a good idea to learn data rates for every destination. Thus, one question is how the sender can choose the proper rates for multiple receivers without losing the advantage of having timely feedback.

A third issue is that multi-destination aggregation will allow aggregating broadcast frames and unicast frames having different destinations into a single transmission, while still supporting size adaptation to find the best aggregation size for the aggregate. However, the size adaptation described in the previous Chapter focused on finding the best size for one destination. Thus, one question is how we can choose the best aggregation size for multi-destination aggregation.

A final issue is that we will have to aggregate multiple packets from a single queue. This extends the design described in the previous Chapter by grouping multiple frames having different destinations into a single transmission. Here, one question is how we can aggregate multiple frames having different destinations without changing the semantics of the frame at the head of queue. This is important because, as the aggregation size becomes small, this makes our aggregation system work the same as the no aggregation system does.

As discussed below, each issue gives rise to a number of design choices. Fully exploring these possibilities is a larger goal than we aim for here. Instead, our goal is to show that the idea of multi-destination aggregation is feasible and worth further exploration. Thus, in the design and implementation below, we often choose the

simplest approach, rather than a more sophisticated one that might possibly result in greater performance.

5.1.1 Queue Management

For our design, we build upon the same design principle as described in Section 4.3.2: frame aggregation should be an optimization and not fundamentally alter the basic semantics of transmission. This design principle will clarify key design decisions about multi-destination aggregation. Thus, as we did in the previous Chapter, we keep using a single queue for both the broadcast and unicast packets incoming from higher layers and aggregate based on this queue.

In keeping with our design principle, each transmission follows the semantics of the frame at the head of queue. Thus, if the frame is a unicast, a sender first sends an RTS that has the same destination as the data. In addition, the contention window size, used for random backoff, is determined by the retry count of the data at the head of the queue. In this case, the basic strategy is that we aggregate all the unicast packets having the same destination as the data. Then, if we still have space in the aggregated frame, we aggregate any broadcast packets and finally unicast packets having different destinations. For our design, the aggregation size is determined by using size adaptation, the details of which will be discussed in Section 5.1.4.

If the frame at the head of the queue is a broadcast, we could choose one of two designs. The first possibility is that we could aggregate broadcast packets only in the aggregated frame. However, this has a disadvantage that we cannot use size adaptation that allows a bigger frame size for an aggregate. A possible solution could be that we use the first unicast frame in the queue to control the aggregate. Thus, we first send an RTS to get the best aggregation size for the destination of the unicast frame. Then, based on the best size, we first aggregate broadcast frames

and, if we still have space in the aggregated frame, then we aggregate unicast frames having the same destination as the first unicast frame in the queue. However, this has a disadvantage of requiring the additional cost of a RTS/CTS exchange by transmitting broadcasts with unicast frames.

This overall approach has two advantages: one, it maintains the semantics of the frame at the head of queue, and two, we can minimize the need to support multiple rates and ACKs by aggregating all the frames having the same destination as the data at the head of queue, then broadcast frames not requiring link-level ACKs, and finally unicasts having different destinations.

5.1.2 Controlling Link-level ACKs

We consider controlling link-level (block) ACKs from multiple receivers. Our design principle makes it clear that a receiver sends a link-level ACK immediately after receiving the frame at the head of queue.

Now we consider the design space for controlling link-level ACKs for other destinations' frames that are not at the head of queue. One end of the design space is that the link-level ACKs are not delayed. Obviously, this design is unlikely to work, as discussed earlier.

The other end of the design space is that the ACKs are delayed indefinitely. Keeping with the idea that this is an optimization, what happens in this case is that when the packet eventually gets to the front of queue, a sender transmits an RTS first before the packet transmission. At this time, the sender transmits an RTS with the flag of "more data" that indicates the sender has more packets to be sent. If the sender has no more data (i.e., the flag is set to 0), a receiver responds to the RTS by sending an ACK. The ACK includes bitmap information for the data that the receiver has already received from the sender. On the other hand, if the sender has more data to be sent (i.e., the flag is set to 1), the receiver responds to the RTS by

sending a CTS. Then, the sender transmits data and finally the receiver responds to the data by sending back an ACK.

In between these two designs, we could utilize an intermediate amount of delay. One way of doing this would be for each receiver to send back an ACK in a slotted approach. A sender reserves the floor for enough time to get back ACKs from multiple receivers, and then each receiver can send back an ACK in turn. Another approach would be that a receiver defers sending an ACK until a random timer expires. However, this approach has the disadvantage that the overhead of sending the ACK is significant when the ACK is sent alone. Thus, another possibility for our design is that the ACK is piggybacked on outgoing packets. This approach will be explored in Chapter 6.

Our implementation includes the version of delayed ACKs using indefinite and intermediate amounts of delay. When a node receives the frames that have different destinations from the frame at the head of queue, it starts a timer for an ACK transmission. Note that the node can know whether a frame is at the head of queue based on where the frame is placed in an aggregate. If the node receives an RTS from the sender, it responds to the RTS by sending an ACK or a CTS. In both the cases, if the node sends back an ACK, as a response for an RTS or data, before the timer expires, it resets the timer. On the other hand, if the timer expires, the node sends back an ACK alone.

5.1.3 Supporting Multiple Rates

We consider supporting multiple rates for multi-destination aggregation. For each destination, we will track our current estimate of the best rate explicitly. Now, the issue becomes how a sender can update that rate. Our design principle makes it clear how this works for the frame at the head of queue. Thus, if we use RBAR, a sender will update the rate based on a RTS/CTS exchange and then use the most

up-to-date rate to send the frame at the head of queue.

Here, a key question is how a sender can update data rates for the unicast frames having destinations different from the frame at the head of queue. We first consider the design space of updating data rates for other destinations at a sender. One possible approach would be using a RTS/CTS exchange. Thus, a sender can use an explicit request when channel information is needed. However, even when a sender does not use an explicit request, it can update data rates for other destinations. One approach would be using reciprocity. In this approach, a sender overhears transmissions by neighboring nodes, and it can update data rates for these nodes. However, one question introduced by this design is if there is some way of deciding whether the channel is reciprocal or not. We decide this based on a RTS/CTS exchange. When the sender has data to transmit, it first transmits an RTS. As a response to the RTS, the sender gets back a CTS carrying the measured SNR from the RTS. At this time, the sender measures the SNR from the CTS, then calculates the gap between the SNRs in both directions, and finally stores this gap for that destination in a table. Thus, whenever the sender overhears data transmissions and the gap for that destination is in the table, it can update the data rate based on the gap and the measured SNR from the transmissions. From a series of experiments, we observed that this gap is stable in our indoor environment. This is because this gap comes from variations in the hardware components that individual radios have.

Now, we consider the receiver side. A receiver keeps track of channel state by overhearing data transmissions by neighboring nodes, and thus it can get the current estimate of the best rates for these nodes. Here, one question is when the receiver should send back channel information. One possible approach would be that a receiver can send back the channel information whenever it overhears data transmissions by neighbor nodes. Thus, whenever the receiver sees data transmissions, it can use that to learn about the channel. Obviously, this is not a good idea

because it causes significant overhead. Thus, in the case that link-level ACKs are sent back, we could reduce overhead by carrying channel information on the ACKs. Further, we would extend this design by piggybacking the channel information on outgoing user packets. This will be explored in Chapter 6. Another approach would be that a receiver can send back channel information when it gets an explicit request. Another approach would be that a receiver keeps track of channel and sends back channel information when the channel changes.

Our implementation includes rate adaptation using reciprocity. Thus, a sender estimates the channel by overhearing data transmissions and compensates for the gap between the SNRs in both directions by using a RTS/CTS exchange.

5.1.4 Supporting Size Adaptation

We consider how to support size adaptation for multi-destination aggregation. In this Section, we discuss possible solutions for various cases and then present our design choice for each case. Performance of each design is highly dependent on traffic and channels, which makes it difficult to find the best design for all possible scenarios. Thus, our design choice is based on which one is the simplest to implement.

Size adaptation, described in the previous Chapter, focused on finding the best size for one destination. However, multi-destination aggregation supports multiple destinations and rates, which might cause incorrect size adaptation in the case where an aggregate has unicast frames having different destinations from the frame at the head of queue and those frames use a higher rate than that for the head of queue. Thus, for size adaptation, we consider broadcasts, unicast frames having the same destination as the frame at the head of queue, and unicast frames having other destinations using the same or lower rates than that for the frame at the head of the queue. This avoids burst drops caused by packets destined for other destinations using a higher rate than that for the frame at the head of queue.

We extend the broadcast aggregation format to support multi-destination aggregation. Thus, in an aggregate, the broadcast portion is located at the front, followed by a series of unicast portions that can include unicast frames having different destinations. The location of each unicast portion for one destination is determined by where the first unicast frame having that destination is placed in the queue. Thus, the unicast portion for the destination of the data at the head of queue precedes the unicast portions for other destinations than head of queue. The details of the aggregation format are described in Appendix C.1.

For our design, if the data at the head of queue is unicast or an aggregate has a unicast frame, a sender always transmits an RTS before sending the aggregate. Thus, based on a RTS/CTS exchange, the sender can keep track of the best size for each destination. Further, for some approaches, we can use this for the sender to find the minimum of the best sizes for all destinations.

Case 1: unicast1+unicast2

We consider the case where unicast is at the head of queue and, after all unicasts having the same destination as the data at the head of queue, the next frame in the queue is a unicast having a different destination from the unicast at the head of queue.

One possible approach would be that a sender uses its own best size for each destination. Thus, the sender first aggregates all unicast frames having the same destination as the data at head of queue based on the best size for the head of queue. Then, the sender searches for the best aggregation size for the unicast frames having a different destination from the data at the head of queue. If the best size is bigger than the aggregated frame size, the sender aggregates the unicast frames having that destination up to the best size.

Another approach would be that a sender uses the best aggregation size for

the head of queue to aggregate all the unicast frames having different destinations. Thus, we first aggregate all the unicasts having the same destination as the unicast frame at the head of queue based on the best size for the head of queue. Then, we aggregate unicast frames having different destinations from the data at the head of queue based on the best size for the head of queue.

Another approach would be that a sender uses the best aggregation size for the head of queue as a maximum bound. Thus, the sender first aggregates all the unicast frames having the same destination as the unicast frame at the head of queue based on the best aggregation size for the head of queue. Then, the sender searches for the best aggregation size for the unicast frames having different destination from the data at the head of queue. If found, the sender aggregates the unicast frames having that destination based on the minimum between the best size for that destination and the best size for the head of queue.

For the purpose of our experiments, we chose the simplest way of using the best aggregation size for the head of queue to aggregate all unicast frames. Thus, based on the best size for the head of queue, we first aggregate the unicast frames having the same destination as the unicast at the head of queue and the unicasts having other destinations than the head of queue.

Case 2: unicast+broadcast

We consider the case where unicast is at the head of queue and, after all unicasts having the same destination as the data at the head of queue, the next frame in the queue is a broadcast.

One possible approach would be that a sender uses the best aggregation size for the destination of the head of queue to aggregate the unicast frames having that destination and then uses the minimum of the best sizes for all destinations to aggregate broadcast frames. Thus, we first aggregate all unicast frames having the

same destination as the unicast frame at the head of queue. Then, if the minimum size is bigger than the aggregated frame size, we aggregate broadcast frames up to the minimum size.

Another approach would be that a sender uses the best aggregation size for the head of queue to aggregate both unicast and broadcast frames. Thus, we first aggregate all the unicasts having the same destination as the unicast frame at the head of queue based on the best size for the head of queue. Now, we consider how to aggregate broadcast frames. One approach would be that we can aggregate all broadcast frames up to the best size for the head of queue. Another approach would be that we can aggregate a fixed number of broadcast frames. The fixed number can be a parameterized value. Another approach would be that we can aggregate a single broadcast frame only. This is reasonable because sending broadcast frames is less reliable than unicast frames and it is difficult for a sender to know the most up-to-date sizes for all destinations.

For the purpose of our experiments, we chose the simplest way of using the best size for the head of queue to aggregate unicast frames and allowing a fixed number of broadcast frames in the aggregate. Thus, we use the best aggregation size for the head of queue to aggregate all unicast frames having the same destination as the data at the head of queue and then, if we still have space, we aggregate a fixed number of broadcast frames. For our experiments, we set the fixed number of broadcast frames to 20.

Case 3: broadcast+unicast

We consider the case where a broadcast is at the head of the queue and, after all broadcasts in the queue, the next frame is a unicast.

One possible approach would be that we can aggregate all broadcasts only based on the minimum of the best sizes for all destinations. Another approach would

be that we can aggregate all broadcasts only based on a fixed aggregation size.

Another approach would be that we search for the first unicast frame in the queue and send an RTS to get the best aggregation size for the destination of the unicast frame. Then, based on the best size, we aggregate all broadcasts, and then, if we still have space, we aggregate unicast frames. Another approach would be that we aggregate a fixed number of broadcast frames and then we aggregate unicast frames up to the best size. Another approach would be that we aggregate a single broadcast frame and then we aggregate unicast frames up to the best size.

For the purpose of our experiments, we chose the simplest way of using unicast frames to control an aggregate and allowing a fixed number of broadcast frames in an aggregate. Thus, a sender transmits an RTS to acquire the best aggregation size for the destination of the first unicast frame in the queue. Then, the sender aggregates a fixed number of broadcast frames and then unicast frames based on the best size.

5.2 Experimental Results

We present performance results for UDP and TCP traffic on both real and emulator-based channels to explore the effectiveness of multi-destination aggregation. For multi-destination aggregation, we used the version of delayed ACKs using infinite and intermediate amounts of delay to control link-level ACKs, the reciprocity-based rate adaptation scheme to support multiple rates, and the size adaptation scheme based on data at the head of queue to find the best aggregation size for an aggregate.

The details of the experimental setup have been presented in Section 2.4. Here, the MAC frame size increases by 4 bytes because the frame format used for our design follows the IEEE 802.11n standard which has a 4-byte prefix field. Thus, a 1KB UDP packet results in 1140B MAC frame and a 1357-byte TCP packet results in a 1468B MAC frame. In addition, we use the same format for block ACKs, as

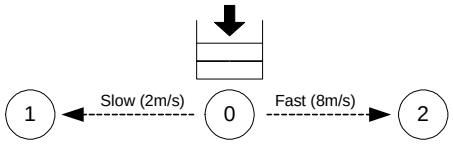


Figure 5.1: Two UDP flows with time-varying channels

described in Appendix B, but the bitmap size increases by 8 bytes to provide more space for the ACKs.

5.2.1 UDP Traffic

We designed these experiments to study the impact of queueing on the performance of multi-destination aggregation. For these experiments, we used UDP traffic because this allows us to change the queueing behavior easily by controlling the packet interval. To begin, we validate our rate adaptation and size adaptation schemes in emulator-based time-varying channels. Then, we present the performance results for no aggregation, single-destination aggregation and multi-destination aggregation in real channels.

Validation of Rate and Size Adaptation in Time Varying Channels

In the presence of time varying channels, we demonstrate our size and rate adaptation schemes. It was difficult to control time varying channels by using real channels, and thus we used emulator-based channels that allow reproducing channels with controllable time variation. We performed an experiment to explore aspects of our rate and size adaptation schemes by measuring the performance and tracing the size and rate variation.

Figure 5.1 shows the topology for our experiments. We used two UDP flows, each of which travels in the opposite direction with a fixed packet interval of 3 seconds. The packets from two flows arrive at the same time, and thus node 0 will

Table 5.1: Performance comparison for single-destination and multi-destination

	Single-Destination	Multi-Destination
Average Throughput^{+}	1.51 Mbps	1.58 Mbps
Average Link Data Rate	4.9 Mbps	4.9 Mbps
Packet Error Rate	0.5%	3.0%

$^{+}$ indicates the throughput for one UDP flow

always have packets having different destinations. In addition, each flow is transmitted over different time varying channels: one is a slow-changing channel having a node speed of 2 m/s and one is a fast-changing channel having a node speed of 8 m/s. Using this topology allows us to demonstrate our size and rate adaptation schemes by tracing which rates and sizes are chosen by our scheme over different time varying channels.

The first question we explored using this topology is how time varying channels have an impact on the overall performance. In addition, we explore whether rate selection using reciprocity is working. We did this by measuring the average throughput, average link data rate, and packet error rate for single-destination and multi-destination aggregation schemes.

Table 5.1 presents the average throughput, average link data rate, and packet error rate for single-destination and multi-destination aggregation schemes. This result shows that, for throughput, multi-destination aggregation achieves some performance gain over single-destination aggregation. This is because multi-destination aggregation allows aggregating packets having different destinations, which improves aggregation efficiency. However, the throughput gain is smaller than our expectation because of increased packet drops. For our size adaptation, our current implementation is to use the best aggregation size for the head of queue to aggregate all unicast frames. This causes occasional packet drops for node 2's data when node 1's data is at the head of queue and node 2's data is aggregated with node 1's data. This is

because node 2's channel is faster than node 1's channel. Thus, in the case where a sender transmits multi-destination packets over different time-varying channels, we could improve the performance by using size adaptation based on the best size for each destination, as we described in Section 5.1.4. In addition, the result shows that, for the average data link rate, single-destination and multi-destination aggregation schemes have the same performance, which means that reciprocity is working.

The second question we explored using this topology is whether our size adaptation scheme is working. Figure 5.2 shows a MAC trace of our size and rate adaptation schemes. The X-axis is the data transmission number. The left Y-axis is the aggregation size in KB, and the right Y-axis is the data rate chosen for each transmission in Mbps. We created this trace based on the same experiment used for measuring the overall performance. A circle shows the aggregation size when node 1's data is at the head of the queue, while a square shows the aggregation size when node 2's data is at the head of queue. An open circle or square shows that the transmission carries multi-destination packets, while a closed circle or square shows that the transmission carries single-destination packets. In addition, a line with point markers shows data rate for node 1 and a line with 'x' markers shows data rate for node 2.

Figure 5.2 shows that most transmissions use an aggregation size of 20KB when node 1's data is at the head of queue, while most transmissions use an aggregation size of 10KB when node 2's data is at the head of queue. This is because the aggregation size is determined by the channel coherence time, and the channel for node 2 changes faster than that for node 1. Thus, when node 1's data is at the head of queue, the aggregation size is large, and all the transmissions carry multi-destination frames (open circles in the graph). On the other hand, when node 2's data is at the head of queue, the aggregation size is small and most transmissions carry single-destination frames only (closed squares in the graph). In addition, we

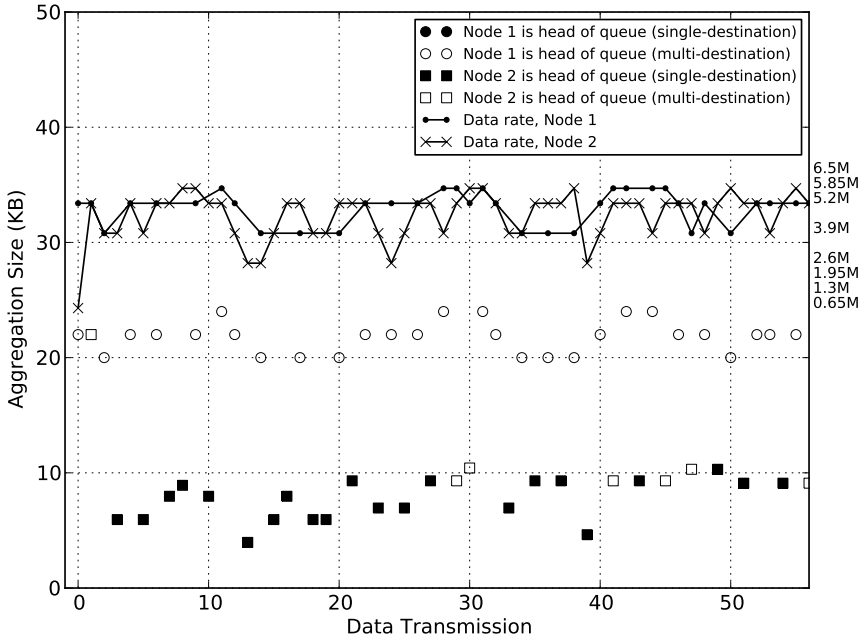


Figure 5.2: MAC trace of size and rate adaptation schemes for multi-destination aggregation

observe that the aggregation size (in bytes) varies with the data rate. This is because our size adaptation scheme converts the aggregation size from physical samples to bytes based on the rate, and thus the aggregation size (in bytes) increases when a higher rate is chosen.

We also observe that, for both the nodes, our rate adaptation scheme chooses data rates between 2.6 Mbps and 5.85 Mbps. One exception is that, for the first transmission, our rate adaptation scheme chooses 0.65 Mbps for node 2. This is because node 1's data is at the head of queue, and thus we can update the rate for node 1 based on a RTS/CTS exchange. On the other hand, at this time, we do not know about the channel for node 2, and thus node 2's data is transmitted at the base rate. However, after this transmission, we observe that our rate adaptation

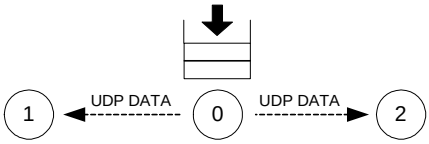


Figure 5.3: Two UDP flows with real channels

scheme updates the data rate for node 2 correctly.

Throughput vs. Packet Interval

We designed these experiments to study the impact of queueing on throughput for no aggregation, single-destination aggregation and multi-destination aggregation. We did this by measuring the throughput as a function of packet interval with a variety of UDP flows. A short packet interval means increased queueing. This measurement will help us to study how queueing effects multi-destination aggregation because we can change queueing behavior by changing the packet interval and by using different traffic characteristics.

Figure 5.3 shows the topology used for our experiments. This is the same as the previous topology except that we used real channels. For these experiments, the reason why we do not use time varying channels is that we focus on the impact of queueing on the throughput. We used two UDP flows, each of which travels toward a different receiver. In addition, we used three different traffic characteristics: two fixed-interval UDP flows, one fixed-interval UDP flow and one fixed-interval UDP flow with jitter, and two poisson UDP flows. Depending on traffic characteristics, we will see different queueing behavior, which allows us to explore the effectiveness of multi-destination aggregation.

Two fixed-interval UDP flows We first consider two fixed-interval UDP flows in which packets having different destinations arrive at the same time. We expect that multi-destination aggregation will outperform single-destination aggregation and no

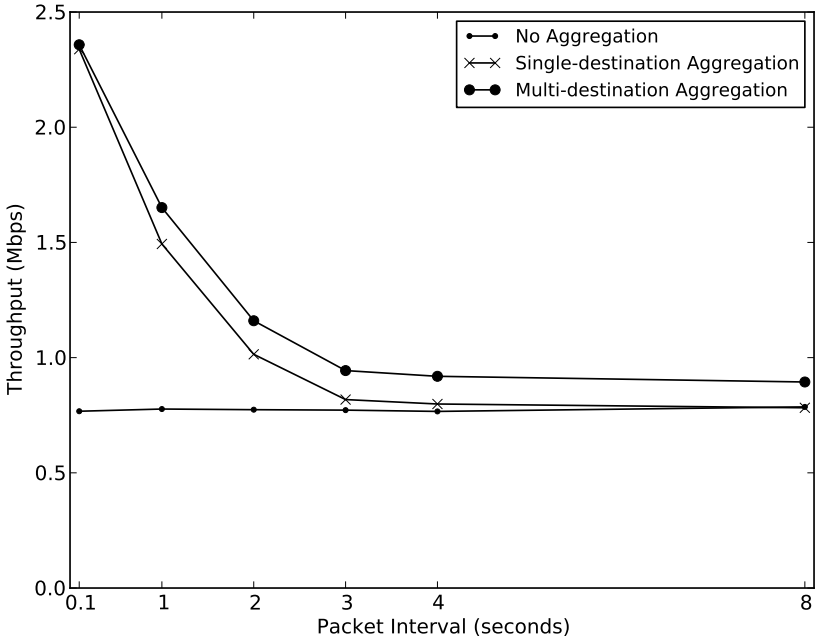


Figure 5.4: Throughput for two fixed-interval UDP flows

aggregation regardless of the packet interval. This is because, in this case, the two flows are synchronous (i.e., packets incoming from two flows arrive at the same time.) and thus we will always have the packets having different destinations, resulting in improved aggregation efficiency for multi-destination aggregation. Figure 5.4 shows the average throughput as a function of packet interval. The X-axis is the packet interval in seconds, and the Y-axis is the average throughput in Mbps.

Figure 5.4 shows that the performance gap between aggregation and no aggregation becomes significant as the packet interval decreases. This is because using a shorter packet interval allows more queueing, resulting in more aggregation. On the other hand, no aggregation can not get any benefit from queueing.

Now, we compare the throughput between single-destination aggregation and multi-destination aggregation. The result shows that multi-destination aggregation

outperforms single-destination aggregation for all the packet intervals except for the interval of 0.1 seconds. This is because, for the very short interval, we have enough queueing for single-destination aggregation to use all the space, and thus multi-destination aggregation behaves the same as single-destination aggregation. However, as the packet interval increases, queueing for single-destination aggregation is less than the aggregation capacity, and thus multi-destination aggregation can achieve some performance gain over single-destination aggregation by allowing aggregation of multiple packets having different destinations. Further, as we expected, the two flows are synchronous, and thus multi-destination aggregation can achieve performance improvement over single-destination aggregation even for very long packet intervals.

One fixed-interval UDP flow and one fixed-interval UDP flow with jitter The next interesting question is what if packets having different destinations do not arrive at the same time. For this experiment, we used one fixed-interval UDP flow and one UDP flow with a interval having some jitter. We created this jitter by using the uniform distribution from $[\frac{-2}{\text{interval}}, \frac{2}{\text{interval}}]$. Figure 5.5 shows the average throughput as a function of packet interval. The X-axis is the packet interval in seconds, and the Y-axis is the average throughput in Mbps.

Figure 5.5 shows that multi-destination aggregation achieves some performance gain over single-destination aggregation for medium packet intervals. This is because, for the medium packet intervals, the service rate is faster than the arrival rate from one flow but less than the sum of the arrival rate from each flow. Thus, there is enough queueing for multi-destination aggregation, resulting in improvement of aggregation efficiency for multi-destination aggregation. However, for a packet interval of 8 seconds, all the schemes have the same performance, which is different from the previous result. This is because this traffic is asynchronous (i.e., packets incoming from two flows do not arrive at the same time.) and thus, for the

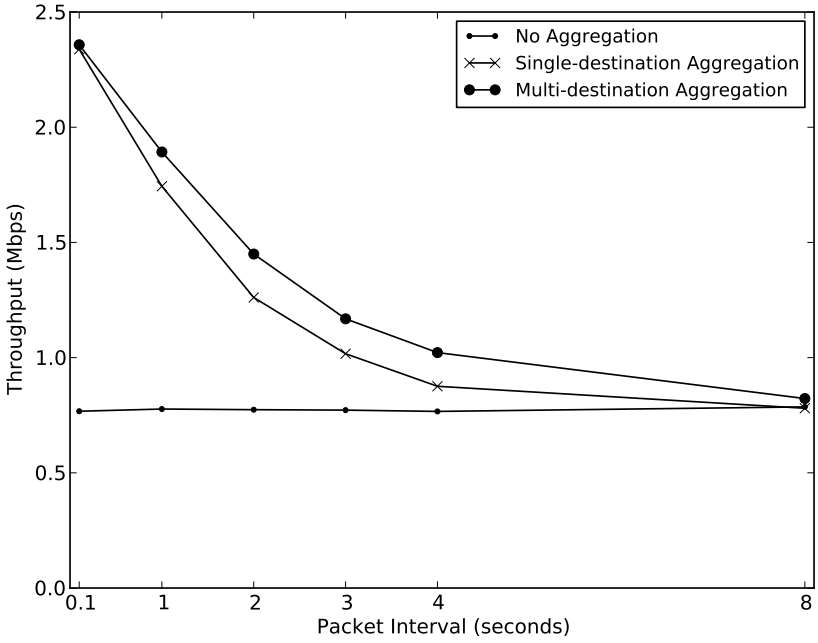


Figure 5.5: Throughput for one fixed-interval UDP flow and one fixed-interval UDP flow with jitter

very long interval, most transmissions carry a single packet only.

Two poisson UDP flows The next interesting question is what if packets arrive in a bursty manner. For this experiment, we used two poisson UDP flows. Figure 5.6 shows the average throughput as a function of average packet interval. The X-axis is the average interval in seconds, and the Y-axis is the average throughput in Mbps.

Figure 5.6 shows that, for an average interval of 8 seconds, aggregation achieves some performance improvement over no aggregation, which is different from the previous results. This is because poisson traffic is bursty and thus there is queueing for aggregation even when the average interval is very long. In addition, for a packet interval of 0.5 seconds, multi-destination aggregation has similar performance to single-destination aggregation though multi-destination aggregation

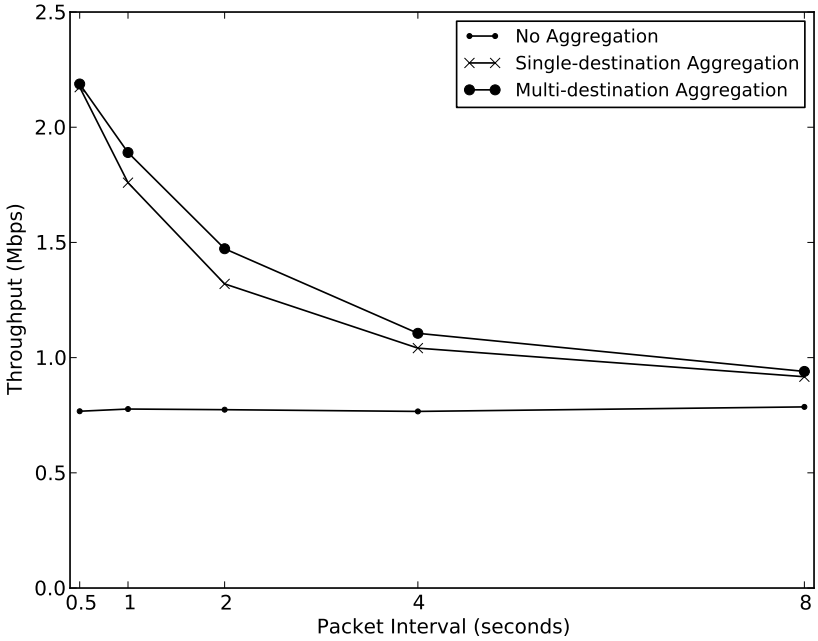


Figure 5.6: Throughput for two poisson UDP flows

has some transmissions that carry multi-destination packets. This is because, in this case, queueing for single-destination aggregation is slightly lower than the aggregation capacity and thus the performance improvement is negligible.

5.2.2 TCP Traffic

It is interesting to study TCP because TCP changes queueing behavior by controlling traffic characteristics based on the network and bandwidth that a system uses. In addition, TCP has multi-destination traffic inherently because it requires the exchange of TCP data and TCP ACKs traveling in the opposite direction. Thus, the goal of these experiments is to study how multi-destination aggregation has an effect on TCP performance for a variety of topologies. We begin by studying the impact of multi-destination aggregation on TCP performance for three basic

topologies. We then present the performance results for our technique with more complex topologies, such as when more relay nodes become involved or when the network becomes congested.

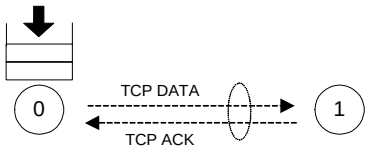
For our experiments, we used "TCP cubic" [1], which improves the scalability of TCP for high-speed networks [42]. All the experiments are performed in real channels, and we set the maximum queue size at the MAC to 100 packets.

TCP Performance for Three Simple Topologies

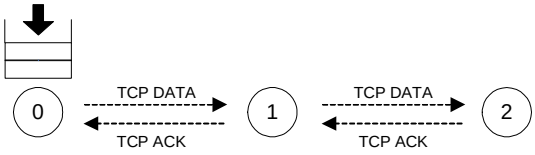
We designed these experiments to study how TCP interacts with multi-destination aggregation for a variety of topologies. We did this by comparing the throughput, TCP congestion window size, MAC frame size, and queue size for no aggregation, single-destination aggregation, and multi-destination aggregation schemes for three different simple topologies.

We first consider the simplest topology having a single TCP flow over a 1-hop network (Figure 5.7(a)). We then extend this topology by adding one more hop (Figure 5.7(b)). Finally, we consider a more complex topology having two TCP flows, each of which travels in the opposite direction over a 1-hop network (Figure 5.7(c)). We expect that, for the 1-hop network, multi-destination aggregation will have the same performance as single-destination aggregation. This is because, in this topology, there is queueing for single-destination aggregation only, and thus multi-destination aggregation behaves the same as single-destination aggregation. We also expect that, for the 2-hop network, multi-destination aggregation will achieve some performance gain over single-destination aggregation. This is because, in this topology, there is a relay node that has queueing for TCP data and TCP ACKs, and multi-destination aggregation can lower overhead by sending TCP

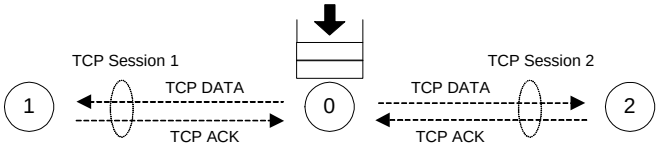
[1] For our experiments, we used TCP cubic because this allows fast increment of the TCP congestion window, and thus we could see that the TCP window size reaches the capacity faster than TCP reno.



(a) A single TCP flow



(b) 2-hop linear topology with one TCP flow



(c) Two TCP flows

Figure 5.7: Three simple topologies

data and TCP ACKs in a single transmission. For the network having two TCP flows, we expect that multi-destination aggregation will show an enhancement over single-destination aggregation. This is because, in this topology, the TCP server has queueing for TCP data having different destinations and, for a given file size, multi-destination aggregation can reduce the number of transmissions by grouping TCP data traveling in the opposite direction into a single transmission.

Table 5.2 summarizes the throughput for no aggregation, single-destination aggregation, and multi-destination aggregation schemes for those topologies. The result shows that, for all the topologies, aggregation achieves significant improvement over no aggregation. This is because aggregation allows sending multiple frames

Table 5.2: Throughput analysis for three simple topologies

	No Aggregation	Single-Destination	Multi-Destination
One TCP flow (1-hop)	1.06 Mbps	4.43 Mbps	4.43 Mbps
One TCP flow (2-hop)	0.52 Mbps	2.27 Mbps	2.34 Mbps
Two TCP flows	0.58 Mbps^{+}	2.15 Mbps	2.15 Mbps

$^{+}$ indicates the average throughput for a single flow

in a single transmission, which improves link-level bandwidth by saving overhead. TCP effectively expands its window to take advantage of aggregation. In addition, for the 1-hop network, single-destination and multi-destination aggregation schemes have the same performance, as expected.

For the 2-hop network, the result shows that the throughput performance for all schemes decreases by half that of the 1-hop network. This is because adding one hop to the 1-hop network decreases link-level bandwidth by half, resulting in a throughput degradation. In addition, as expected, multi-destination aggregation has some performance gain over single-destination aggregation. We will discuss a more detailed analysis in the following Subsection.

For the topology having two TCP flows, the result shows that multi-destination aggregation has the same performance as single-destination aggregation, which is different from our expectation. This is because, in this topology, we observed some transmissions carrying TCP data having different destinations, but the ratio of aggregation of TCP data was small, resulting in negligible improvement. We will discuss more details of this in the following Subsection.

Although Table 5.2 shows the basic performance characteristics of our system, more detailed tracing can give us more insight about the performance of no aggregation, single-destination aggregation, and multi-destination aggregation schemes for those topologies. We created these traces based on the same experiments used for the throughput analysis.

One TCP flow over a 1-hop network The goal of this analysis is to gain insight about how aggregation has an impact on TCP performance when a 1-hop network is used. We did this by tracing the congestion window size, MAC frame size and queue size. We expect that single-destination and multi-destination aggregation will have a similar trace of congestion window size. This is because, in this topology, multi-destination will behave the same as single-destination aggregation.

Figure 5.8 shows traces of congestion window size, MAC frame size, and queue size over 1-hop network for no aggregation, single-destination aggregation, and multi-destination aggregation (from top to bottom). For each scheme, the top graph shows traces of the TCP congestion window size (in segments) and MAC frame size (in Kbytes) as a function of time (in seconds), and the bottom graph shows a trace of queue size (in Kbytes) as a function of time. All the traces are measured at the TCP server (node 0 in Figure 5.7(a)).

The result shows that, as we expected, multi-destination aggregation has a similar trace of congestion window size to single-destination aggregation. In addition, compared to no aggregation, the aggregation schemes improve the TCP capacity significantly and allow fast increment of the congestion window size until reaching the capacity. Further, the trace of MAC frame size shows that, for aggregation, most of transmissions exploit the maximum aggregation capacity. This shows that TCP effectively expands its window to take advantage of aggregation.

One TCP flow over a 2-hop linear network The goal of this analysis is to gain insight about how multi-destination aggregation impacts TCP performance when a 2-hop linear topology is used. We did this by tracing congestion window size, MAC frame size and queue size. Figure 5.7(b) shows the 2-hop linear topology used for this tracing. In this topology, there is a single relay node (node 1) that can have TCP data and TCP ACKs traveling in the opposite direction. We expect that multi-destination aggregation will allow faster increment of TCP congestion window

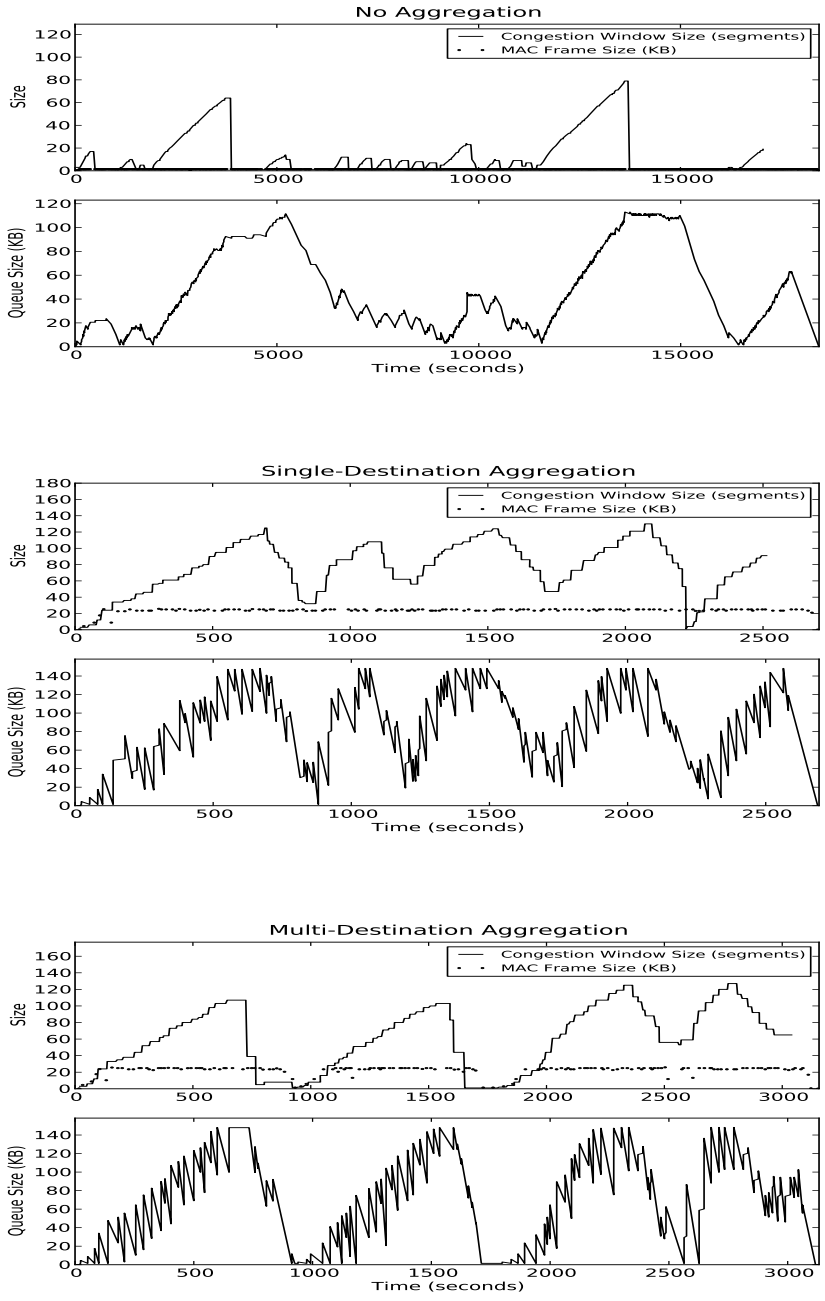


Figure 5.8: Congestion window size, MAC frame size, and queue size for a single TCP flow over a 1-hop network

over single-destination aggregation. This is because multi-destination aggregation allows aggregation of TCP data and TCP ACKs, which lowers overhead by reducing the number of transmissions.

Figure 5.9 shows traces of congestion window size, MAC frame size and queue size over 2-hop linear network for no aggregation, single-destination aggregation, and multi-destination aggregation (from top to bottom). For each scheme, the top graph shows traces of TCP congestion window size (in segments) and MAC frame size (in Kbytes) as a function of time (in seconds), and the bottom graph shows a trace of queue size (in Kbytes) as a function of time. The trace of congestion window size is measured at the TCP server (node 0 in Figure 5.7(b)), and the traces of queue size and MAC frame size are measured at the relay node (node 1 in Figure 5.7(b)).

The result shows that, compared to no aggregation, aggregation allows fast increment of TCP congestion window size, similar to the previous result. In addition, the trace of MAC frame size shows that single-destination aggregation has more transmissions having a small size than multi-destination aggregation. This is because the relay node has queueing for TCP data and TCP ACKs, but single-destination aggregation can aggregate TCP ACKs or TCP data only. On the other hand, multi-destination aggregation allows TCP ACKs to be aggregated with TCP data traveling in the opposite direction. Further, TCP ACKs are small, and thus they can be aggregated with TCP data even when we have not enough space for TCP data. This allows more transmissions to carry TCP data and TCP ACKs, resulting in a decrement in the number of small-size transmissions.

Our analysis shows that the ratio of aggregation of TCP data and TCP ACKs is 23%, which increases the average frame size for multi-destination aggregation. The average size for multi-destination aggregation is 14.7KB, while the size for single-destination aggregation is 13.3KB. As a result, it reduces the number of transmissions for multi-destination aggregation scheme (as a percentage of that for

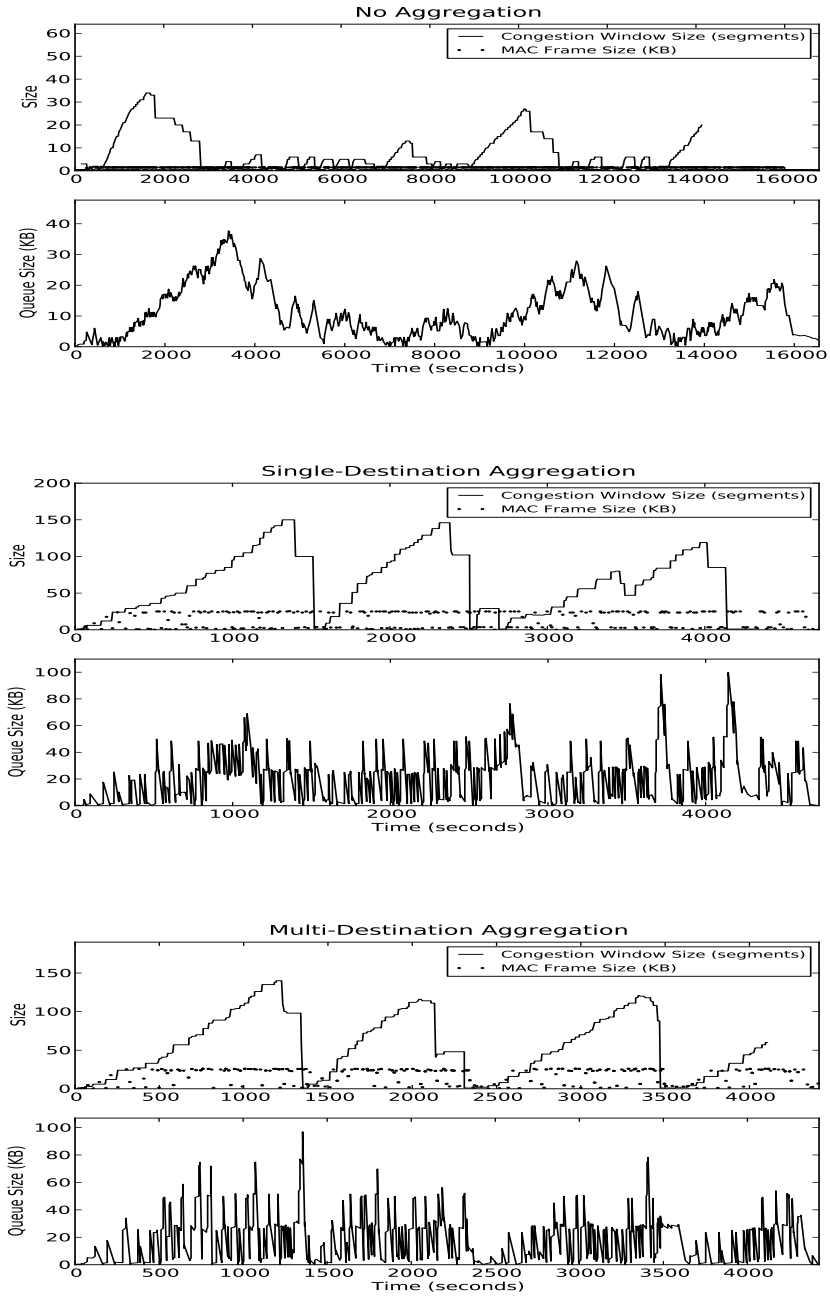


Figure 5.9: Congestion window size, MAC frame size, and queue size for a single TCP flow over a 2-hop network

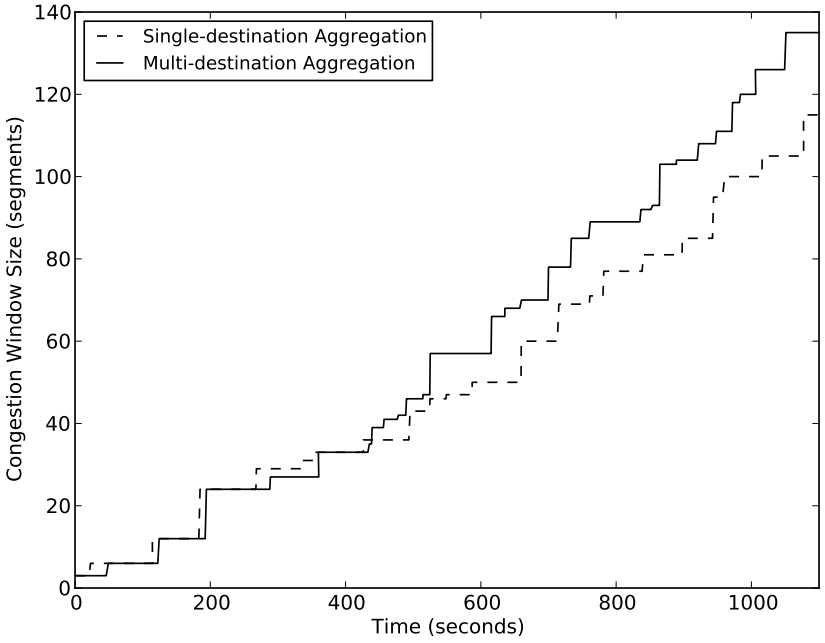


Figure 5.10: Comparison of increment of congestion window size (TCP server)

single-destination aggregation scheme) to 84.7%.

Although this trace gave us an insight about how TCP works with multi-destination for the 2-hop network, it was difficult to compare the increment of congestion window size for single-destination and multi-destination aggregation schemes. Figure 5.10 shows congestion window sizes for single-destination and multi-destination aggregation schemes for the first 1100 seconds. The X-axis is time in seconds, and the Y-axis is the congestion window size in segments.

The result shows that single-destination and multi-destination aggregation schemes have the same increment of congestion window size for the first 500 seconds. However, after that time, multi-destination aggregation increases the congestion window size faster than single-destination aggregation, as we expected. This is because initially the congestion window size is small, and thus it does not exceed

the aggregation capacity. Thus, a single transmission makes its own queue empty, which does not allow TCP data and TCP ACKs to be in the queue simultaneously. However, as the congestion window size increases, a single transmission cannot make its own queue empty, which increases the probability that the relay node has both TCP data and TCP ACKs in the queue. Thus, multi-destination allows aggregation of TCP data and TCP ACKs, resulting in faster increment of congestion window size than single-destination aggregation.

Two TCP flows The goal of this analysis is to gain insight about how multi-destination aggregation has an impact on TCP performance when there are two flows in the network. We did this by tracing congestion window size, MAC frame size and queue size. Figure 5.7(c) shows the topology, which has two TCP flows traveling in the opposite direction over 1-hop.

Figure 5.11 shows traces of congestion window size, MAC frame size, and queue size at the MAC for no aggregation, single-destination aggregation, and multi-destination aggregation (from top to bottom) when there are two TCP flows. For each scheme, the top graph shows traces of TCP congestion window size (in segments) and MAC frame size (in Kbytes) as a function of time (in seconds), and the bottom graph shows a trace of queue size (in Kbytes) as a function of time. All the traces are measured at the TCP server (node 0 in Figure 5.7(c)).

The results show that, for single-destination aggregation, both the sessions begin by growing congestion windows similarly but, at some point, session 2 uses the entire network capacity until completing its own file transfer, which causes a significant fairness problem. This is because a large congestion window size for one session causes a huge amount of delay, caused by queueing and transmission time, for the other session. This results in decreasing the congestion window size because of frequent timeouts. On the other hand, for no aggregation and multi-destination aggregation, we observe that each session shares the network resource fairly. This

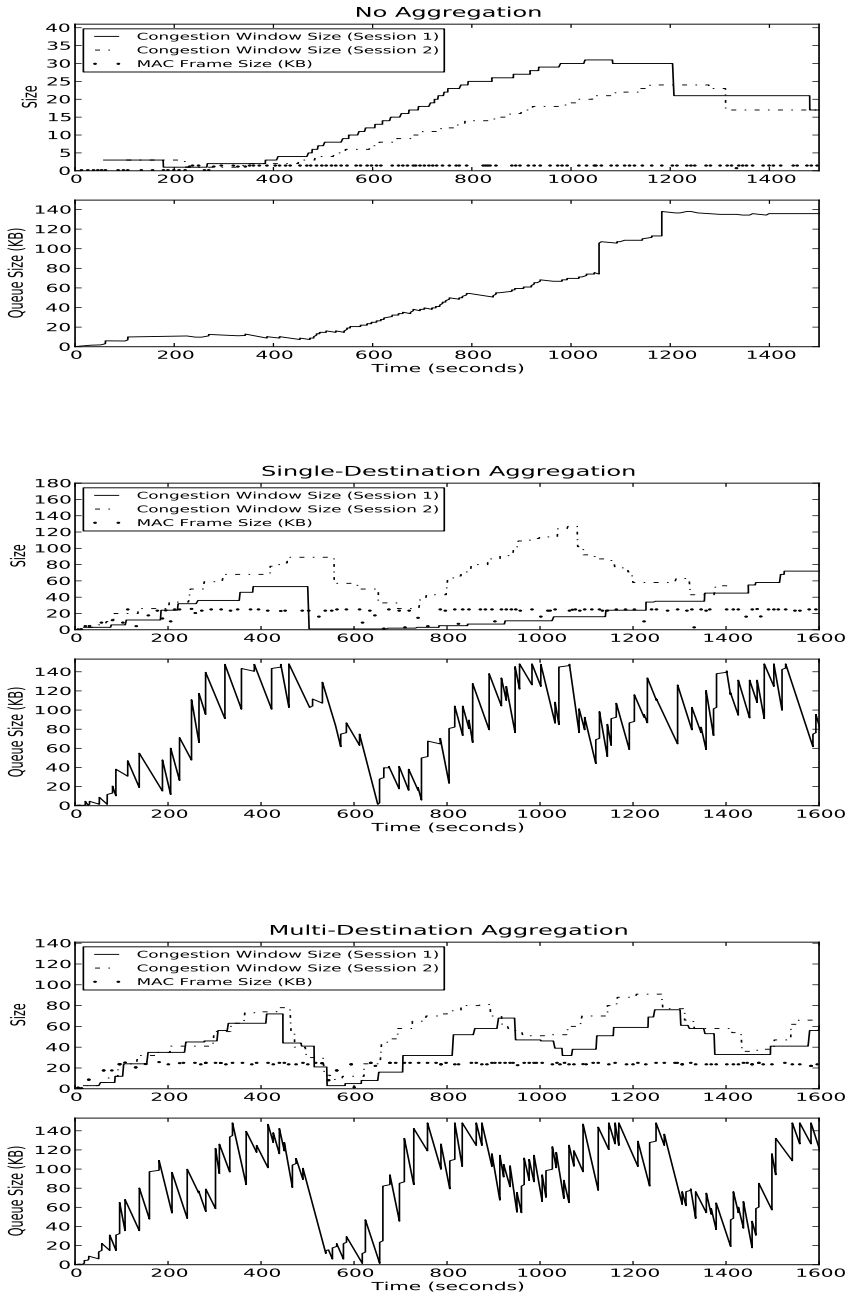


Figure 5.11: Congestion window size, MAC frame size, and queue size for two TCP flows over 1-hop network

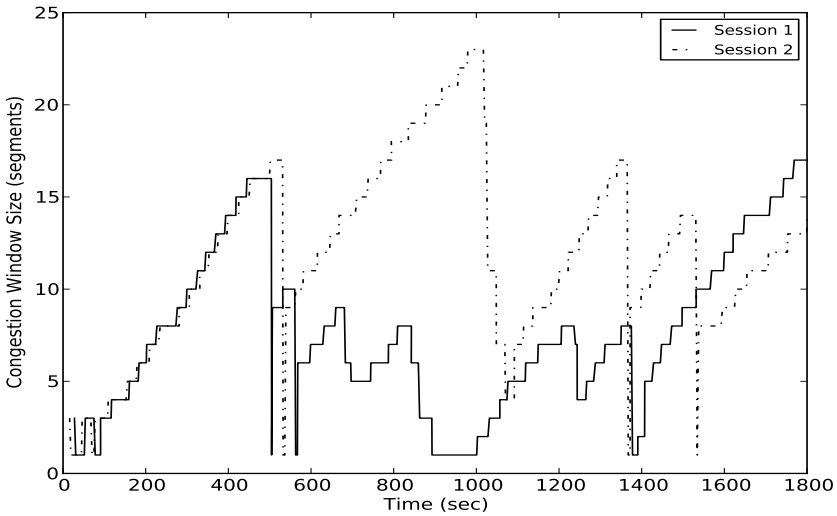


Figure 5.12: Trace of congestion window size for two TCP flows over 1-hop network for multi-destination aggregation

is because, for no aggregation, the packet transmission time is relatively short, and thus each transmission does not have a significant impact on the other sessions' RTT. In addition, multi-destination aggregation can aggregate TCP data having different destinations, which prevents one session's congestion window size from being degraded significantly. The result shows that, for multi-destination aggregation, the congestion window sizes for both the sessions increase to about 70 segments, which is half the congestion window for multi-destination aggregation in the 1-hop network. However, our analysis shows that the ratio of aggregation of TCP data at the TCP server is 9%, which is lower than that of TCP data and TCP ACKs at the relay node in the 2-hop network. As a result, compared to the 2-hop network, the number of transmissions for the multi-destination aggregation scheme (as a percentage of that for single-destination aggregation scheme) increases to 88.5%. This is because, compared to TCP ACKs, TCP data is relatively large, and thus it decreases the probability that we can aggregate TCP data having different destinations.

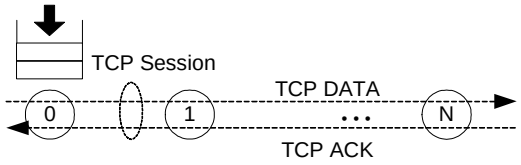


Figure 5.13: Linear topology with one TCP flow

In the previous trace, it seems that multi-destination aggregation has a TCP synchronization problem [43]. To confirm or deny this, we did another set of experiments to trace the congestion window size for multi-destination aggregation. Figure 5.12 shows trace of congestion window size (in segments) for multi-destination aggregation as a function of time (in seconds). This trace shows that multi-destination aggregation does not have a TCP synchronization problem, while still keeping fairness for multiple sessions. We believe that the previous result just coincidentally appeared to synchronization.

Linear Topology

We have looked at the impact of multi-destination aggregation on TCP performance for three simple topologies. We extend our experiments to more complex topologies, such as when more relay nodes become involved or when the network becomes congested. The goal of this experiment is to study the impact of hop count on the performance of multi-destination aggregation. We did this by measuring the throughput of no aggregation, single-destination aggregation, broadcast aggregation, and multi-destination aggregation as a function of the number of hops. Figure 5.13 shows the linear topology used for this experiment. In this topology, more relay nodes become involved. Thus, it is interesting to see how the relay nodes have an effect on the performance of our system. We expect that multi-destination aggregation will show an enhancement over single-destination aggregation due to the increased hop count. This is because increasing hop count means that more

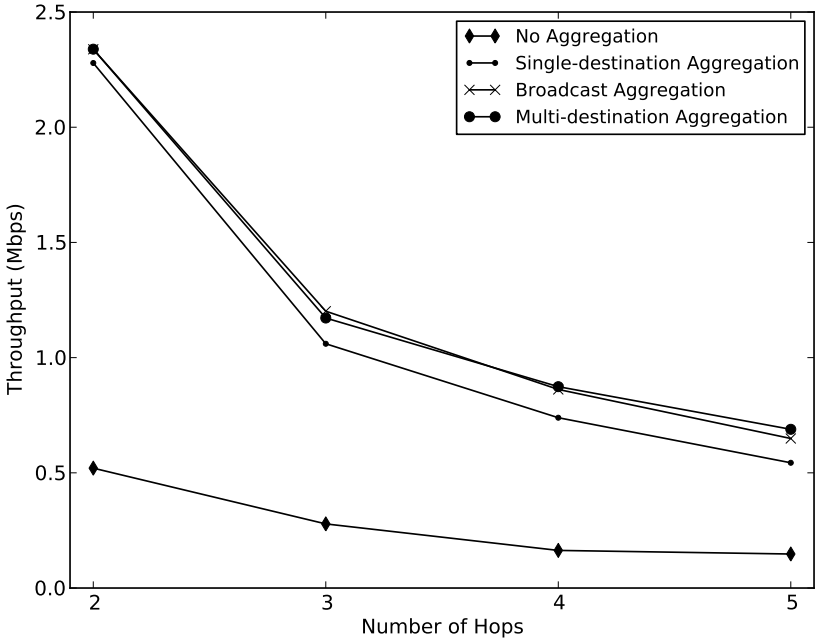


Figure 5.14: Throughput as a function of number of hops

nodes are involved in the aggregation of TCP data and TCP ACKs, resulting in improvement in the aggregation ratio.

Figure 5.14 shows the throughput as a function of the number of hops. The X-axis is the number of hops, and the Y-axis is the end-to-end throughput in Mbps. For the performance comparison, we measured the throughput for broadcast aggregation, described in Chapter 3. For the broadcast aggregation, we needed to decide the data rate for broadcast frames and, for a given channel, we chose a reliable rate of 5.85 Mbps for broadcast frames. In addition, we used rate adaptation using a RTS/CTS exchange to determine the unicast frame rate.

The result shows that aggregation achieves significant improvement over no aggregation. In addition, we observe that multi-destination aggregation has similar performance to broadcast aggregation, which means that multi-destination aggre-

Table 5.3: Performance analysis for linear topology

Number of Hops	2	3	4	5
Frame Size (Single-Destination Aggregation)	13.3KB	6.7KB	5.0KB	4.2KB
Frame Size (Multi-Destination Aggregation)	14.7KB	9.0KB	8.3KB	8.1KB
Aggregation Ratio (Total) (%)	23.0	20.0	21.6	23.2
Aggregation Ratio (Relay Nodes) (%)	52.4	(30.5, 35.6)	(26.7, 31.5, 32.6)	(27.6, 27.3, 40.0, 27.3)
Total TXs (%)	84.7	80.5	64.3	54.3
Performance Gap (%)	3.1	10.3	17.6	27.8

gation achieves the same advantage as broadcast aggregation. Further, as expected, the performance gap between single-destination aggregation and multi-destination aggregation increases as the hop count increases. This is because, as more relay nodes become involved in the aggregation of TCP ACKs and TCP data, the overall aggregation ratio increases. In addition, the increased hop count decreases the network capacity, which results in decreasing the average transmission size. Thus, as we described in Section 4.2.1, the overhead improvement becomes significant as the frame size decreases.

Table 5.3 presents the performance analysis for single-destination and multi-destination aggregation schemes as a function of the number of hops. The frame size indicates the average frame size for all nodes. The aggregation ratio (total) represents the number of multi-destination transmissions at relay nodes divided by the number of total transmissions, while the aggregation ratio (relay nodes) is the number of multi-destination transmissions divided by the number of transmissions at each relay node. In addition, the total TXs represents the number of transmissions for multi-destination aggregation scheme as a percentage of that for single-destination aggregation scheme. The performance gap means the throughput gap between multi-destination aggregation and single-destination aggregation schemes.

This analysis shows that, for both the schemes, the average frame size decreases as the hop count increases. For better understanding, we traced queue size at the TCP server and we found that, for both the schemes, the queue size decreases as the hop count increases, resulting in a decrement of the average frame

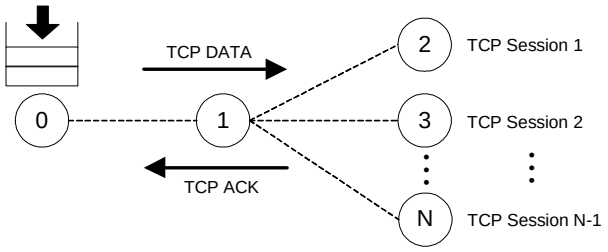


Figure 5.15: Star topology with multiple TCP flows

size. However, the increased multi-destination aggregation ratio increases the gap of the frame size between single-destination and multi-destination aggregation schemes as the hop count increases. Thus, it decreases the number of total transmissions for multi-destination aggregation. The maximum performance gap increases to 27.8% when the hop count is 5.

Now, we compare the throughput between multi-destination aggregation and broadcast aggregation. The result shows that multi-destination aggregation has similar performance to broadcast aggregation. Both the schemes can aggregate TCP data with TCP ACKs traveling in the opposite direction. However, there are tradeoffs between multi-destination and broadcast aggregation. Broadcast aggregation has an advantage of saving link-level bandwidth by ignoring control frames for TCP ACKs. On the other hand, multi-destination aggregation has advantages of allowing rate adaptation and reliable transmissions for TCP ACKs. These tradeoffs result in similar performance between these schemes.

Star Topology

The goal of this experiment is to study the impact of network congestion on the performance of multi-destination aggregation. We did this by measuring the throughput of no aggregation, single-destination aggregation, broadcast aggregation, and multi-destination aggregation as a function of the number of TCP sessions. Figure 5.15 shows the star topology used for this experiment. In this topology, we

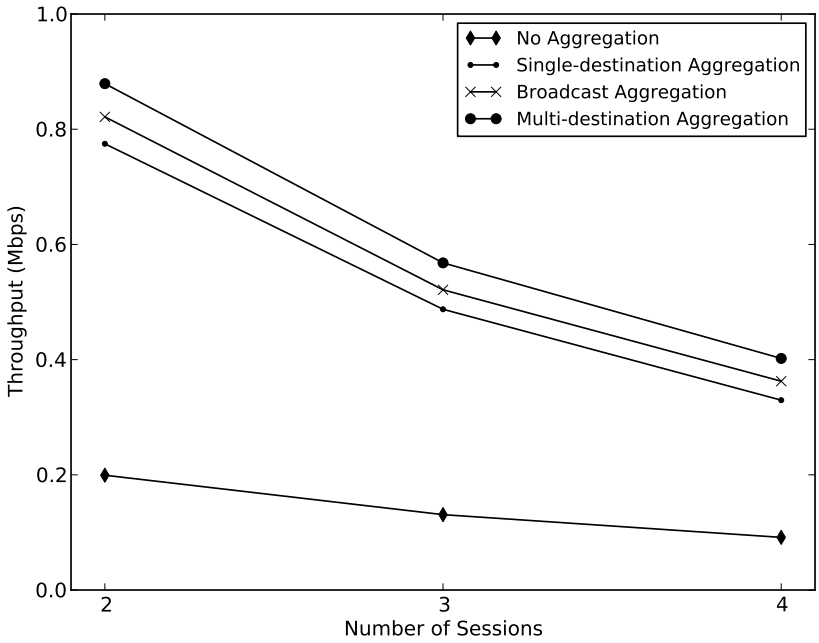


Figure 5.16: Throughput as a function of number of sessions

expect more congestion because the central node (node 1) will be a bottleneck for the TCP streams. Thus, it is interesting to see how network congestion has an effect on the performance of our system. We expect that the performance gap between single-destination and multi-destination aggregation will increase as the number of sessions increases. This is because network congestion causes more queueing at the central node, and multi-destination aggregation allows aggregating frames flowing in all directions.

Throughput Figure 5.16 shows the throughput as a function of the number of TCP sessions. The X-axis is the number of sessions, and the Y-axis is the average throughput for a single flow in Mbps. For broadcast aggregation, we used the same configuration as the linear topology.

The result shows that aggregation achieves significant improvement over no aggregation. In addition, as expected, the performance gap between single-destination aggregation and multi-destination aggregation increases as the number of sessions increases. This is because network congestion leads to longer queues. This allows multi-destination aggregation to have more aggregation opportunities than single-destination aggregation. This is because, in this topology, multi-destination aggregation can aggregate frames traveling in all directions, while single-destination aggregation only supports aggregating frames being transmitted to one destination. In addition, we observe that broadcast aggregation shows an enhancement over single-destination aggregation, while it has worse performance than multi-destination aggregation. This is because, in this topology, broadcast aggregation only supports aggregating TCP ACKs and TCP data having one destination.

Fairness We consider how multi-destination aggregation has an impact on fairness. We examine this by calculating the fairness based on the same experiment as for the throughput measurement. Figure 5.17 shows the fairness index as a function of the number of TCP sessions. For this calculation, we used the fairness index, proposed by Vardalis [44], which reflects the system fairness accurately when the number of TCP sessions is small. The X-axis is the number of sessions, and the Y-axis is the fairness index ranging from 0 to 1.

The result shows that no aggregation achieves improved fairness over aggregation. This is because, for no aggregation, packet transmission time is relatively short, and thus each transmission does not have a significant impact on other sessions' RTT variances. On the other hand, for aggregation, fast increment of the congestion window and long packet transmission times increases the RTT variances for other sessions, resulting in frequent timeouts. In addition, we observe that multi-destination aggregation shows an enhancement over broadcast aggregation and single-destination aggregation. This is because multi-destination aggregation

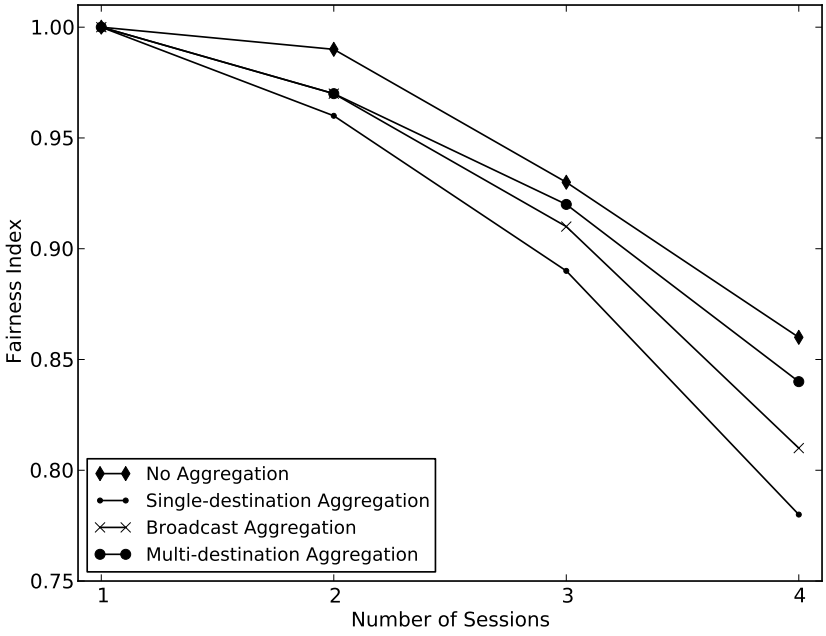


Figure 5.17: Fairness as a function of number of sessions

allows aggregating frames flowing in all directions, which reduces the impact of aggregation on the RTT variances.

Performance analysis Table 5.4 presents the performance analysis for single-destination and multi-destination aggregation schemes as a function of number of sessions. For this analysis, we used the same definitions as we did in linear topology with one exception: the aggregation ratio (central node) represents the number of multi-destination transmissions divided by the number of total transmissions at the central node.

This analysis shows that, for single-destination aggregation, the average frame size decreases significantly as the number of sessions increases. On the other hand, for multi-destination aggregation, the average frame size is not very sensi-

Table 5.4: Performance analysis for star topology

Number of Sessions	1	2	3	4
Frame Size (Single-Destination Aggregation)	13.3KB	8.4KB	6.7KB	6.0KB
Frame Size (Multi-Destination Aggregation)	14.7KB	10.3KB	10.0KB	9.1KB
Aggregation Ratio (Total) (%)	23.0	25.8	26.2	28.9
Aggregation Ratio (Central Node) (%)	52.4	60.6	62.1	70.0
Total TXs (%)	84.7	73.2	62.1	62.0
Performance Gap (%)	3.1	14.2	18.8	21.2

tive to the number of sessions, resulting in the increased gap of the average frame size. This is because, as the number of sessions increases, network congestion becomes significant, and thus the multi-destination aggregation ratio increases. This analysis shows that the multi-destination aggregation ratio at the central node increases significantly as the number of sessions increases. Thus, for a given file size, multi-destination aggregation reduces the number of transmissions. The maximum throughput gap increases to 21.2% when the number of sessions is 4.

Figure 5.18 shows the number of transmissions as a percentage of total transmissions at the central node as a function of the number of destinations. The X-axis is the number of destinations, and the Y-axis is the percentage of transmissions. We created this histogram based on the same experiment as for the throughput measurement. For this analysis, we considered that the number of sessions is 4 and thus, for multi-destination aggregation, an aggregate can include up to five different destinations (i.e., four destinations for TCP data and one destination for TCP ACKs).

The analysis shows that, at the central node, many of transmissions carry an aggregate having two or more destinations. The percentage of transmissions having two or more destinations is 70%. The increased aggregation ratio reduces the impact of transmissions for one session on RTT variances for the other sessions. This allows fair increment of congestion windows for each session, resulting in improved fairness.

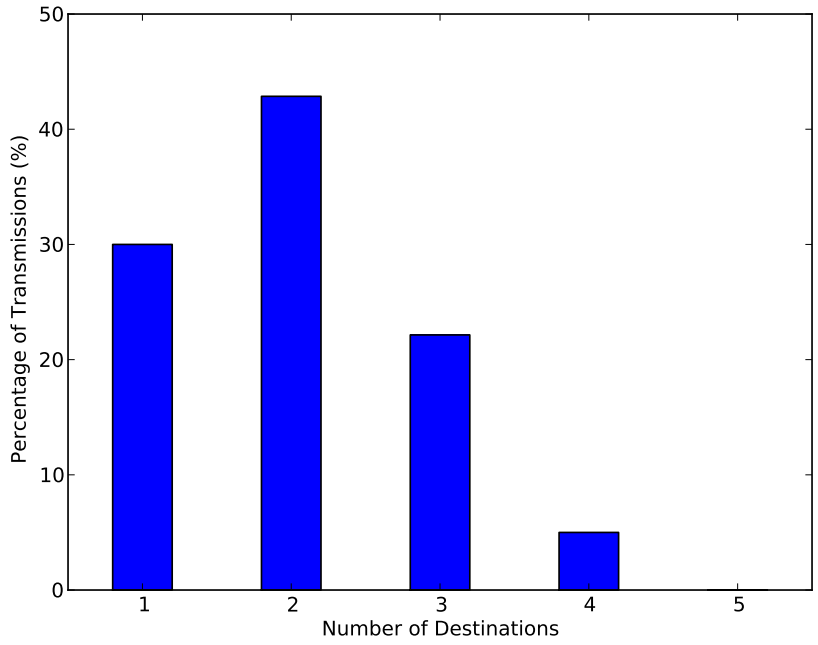


Figure 5.18: Percentage of transmissions as a function of the number of destinations for multi-destination aggregation (the number of sessions=4)

Throughput on the presence of flooding We consider how multi-destination aggregation has an impact on network performance on the presence of flooding. We examine this by measuring the throughput of single-destination aggregation, broadcast aggregation, and multi-destination aggregation as a function of the flooding interval. To simulate flooding, we used the same configuration as described in Section 3.4.3. For this experiment, we used the star topology with two TCP sessions. We used the star topology because the center node is a bottleneck for TCP streams, and it is interesting to see how multi-destination aggregation has an impact on network congestion on the presence of flooding. We expect that the performance gap between single-destination and multi-destination aggregation schemes will increase as the flooding interval decreases. This is because multi-destination aggregation

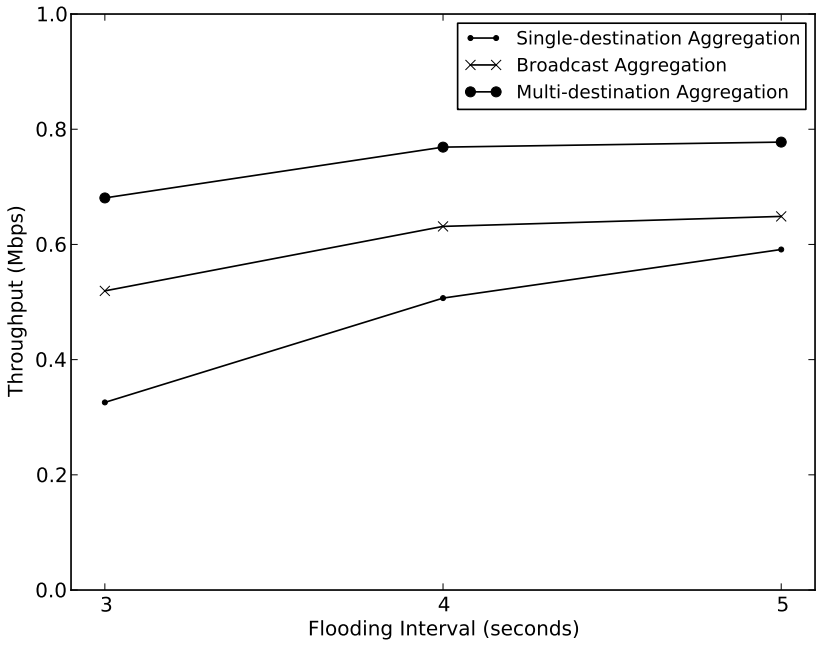


Figure 5.19: Throughput as a function of flooding interval

supports aggregating broadcast frames and unicast frames flowing in all directions, which reduces the impact of flooding on network performance.

Figure 5.19 shows the throughput as a function of the flooding interval. The X-axis is the interval of flooding frames in seconds, and the Y-axis is the average throughput for a single flow in Mbps. For all the aggregation schemes, we set the maximum number of broadcast frames to be aggregated in a single transmission to 20.

The result shows that, as expected, multi-destination aggregation achieves significant improvement over single-destination aggregation. This is because multi-destination aggregation supports aggregating flooding packets, TCP ACKs, and TCP data flowing in all directions, which effectively reduces the overhead of flooding. In addition, we observe that broadcast aggregation shows an enhancement

Table 5.5: Performance analysis for star topology on the presence of flooding

Flooding Interval (seconds)	3	4	5
Frame Size (Single-Destination Aggregation)	1.7KB	2.2KB	2.4KB
Frame Size (Multi-Destination Aggregation)	4.2KB	4.3KB	4.3KB
Total TXs (%)	36.7	50.0	57.1
Performance Gap (%)	106.1	51.0	32.2

over single-destination aggregation, while it has worse performance than multi-destination aggregation. This is because, at the center node, broadcast aggregation can aggregate flooding packets, TCP ACKs, and TCP data having one destination.

Table 5.5 presents the performance analysis for single-destination and multi-destination aggregation schemes as a function of flooding interval. This analysis shows that, for single-destination aggregation, the average frame size decreases as the flooding interval decreases. On the other hand, for multi-destination aggregation, the average frame size is not very sensitive to the flooding interval, resulting in the increased gap of the average frame size. This is because single-destination aggregation supports aggregating flooding packets or unicast frames having one destination, and using a short flooding interval increases the RTT variance for each session, resulting in decreasing TCP transmission rate. On the other hand, multi-destination aggregation reduces the impact of flooding on TCP performance by aggregating frames flowing in all directions. Thus, for a given file size, multi-destination aggregation decreases the number of transmissions. The maximum throughput gap increases to 106.1% when the flooding interval is 3 seconds.

Chapter 6

Metadata Piggybacking

Metadata is information that may be useful to transport but that is not part of the data that is actually being conveyed by the network. Transmitting the delayed ACKs used to support multi-destination aggregation is an example of metadata piggybacking. As a second example, we can consider transmitting channel information for rate adaptation. The channel information is metadata. As a third example, we can consider a MAC protocol where we might broadcast a node's queue state information. The queue state information is metadata.

In networking systems, metadata is important because it provides some information to neighbors that results in improving network performance by exploiting this information for data transmissions. However, the overhead of transmitting metadata limits the possible performance improvement. Thus, taking advantage of the fact that wireless transmissions are broadcast allows us to send metadata at very low cost by piggybacking it with user packets, which means that our potential performance improvement is increased.

6.1 Design Issues and Solutions

The goal of this design is to develop a transmission mechanism that allows piggybacking of metadata with user packets.

From the design point of view, it is obvious that metadata will be aggregated with user packets. Basically, we use multi-destination aggregation, which allows a system to aggregate both broadcast and unicast subframes having different destinations into a single physical frame. Based on this format, metadata can be assembled into the broadcast or unicast portion.

One design issue is whether metadata is useful for all destinations, a group of destinations, or one destination. For example, an ACK is useful only for one receiver. On the other hand, queue state information could be useful for all of the neighbors or a group of the neighbors. Thus, if the metadata is useful only for one receiver, it can potentially be part of the unicasts in an aggregate. If the metadata is useful for all of the neighbors, it can potentially be part of the broadcasts in an aggregate. If the metadata is useful for a group of neighbors, it can potentially be part of the unicasts or broadcasts in an aggregate. This is because transmissions are broadcast and a single transmission will potentially be received by all neighbors. Thus, a group of the neighbors, who are interested in the metadata, could get the information regardless of the type of frame.

Another design issue is how we can aggregate metadata with user packets. For our design, we introduce the design principle that sending metadata is always optional. In keeping with our design principle, our transmission mechanism should allow piggybacking of metadata without limiting aggregation of user packets.

6.2 Working Examples

We evaluated our design using two examples, each of which implemented a transmit mechanism for piggybacking of metadata with user data using Hydra.

First, we coordinate delayed ACKs from multiple receivers using metadata piggybacking. The delayed ACK is metadata because it provides information about the received data frames but is not data itself. In general, the delayed ACK is small, and thus the overhead of sending a delayed ACK alone is significant. Thus, if a node has user packets and a delayed ACK, it aggregates the delayed ACK with the user packets. In this case, the node assembles the delayed ACK into the unicast portion because the delayed ACK is only useful for one node. This can lower the overhead of sending the delayed ACK by piggybacking this ACK onto an existing transmission. This is likely to work because it is often the case that, if one node communicates information to another node, there is information coming back to the first node. An obvious example is a TCP flow that has TCP data and TCP ACKs flowing in the opposite direction. In this case, there will be plenty of data flowing in the proper directions.

Second, we consider supporting multiple rates for multi-destination aggregation. One possible solution was that a node, overhearing other nodes' transmissions, sends the channel information explicitly. However, channel information is small, and thus the overhead of sending channel information alone is significant. Thus, if a node has user packets and channel information, then it piggybacks the channel information with user packets. In this case, this node assembles the channel information into the unicast portion because the channel information is only useful for one sender. This allows the node to send the channel information at very low cost.

6.3 Design

As concrete examples, we chose to implement a transmit mechanism for piggybacking of the delayed ACKs and the channel information with user packets.

When delayed ACKs or channel information need to be sent, a node first constructs a unicast frame based on the destination address. This frame can include the payload of bitmap information for the delayed ACK or SNR for the channel information. To decode this frame correctly, we define a new subframe type for each metadata by using the reserved values in the 802.11 MAC frame format [1].

For piggybacking metadata on user frames, we first aggregate all the frames having the same destination as the frame at the head of queue. Then, if we still have space, we search the queue to find whether there is metadata having that destination. If found, the metadata is aggregated with those frames. Note that an aggregate can include both channel information and delayed ACKs with user frames. Then, if we still have space, we do the same process for the frames having different destinations. Further, if the channel is static and thus we use 2% overhead bound for an aggregate as we did in Chapter 4, we place metadata beyond the 2% overhead bound to aggregate the metadata with user frames.

6.4 Experimental Results

We present performance results for UDP and TCP traffic on real channels to explore the effectiveness of metadata piggybacking. For our experiments, we used multi-destination aggregation with two different rate adaptation schemes: one using reciprocity and one using explicit feedback, as described in Section 5.1.3. For our rate adaptation using explicit feedback, a receiver sends back channel information when the channel changes or an RTS is received. Thus, the receiver keeps track of the channel for all destinations and stores the current estimate of the best rate for

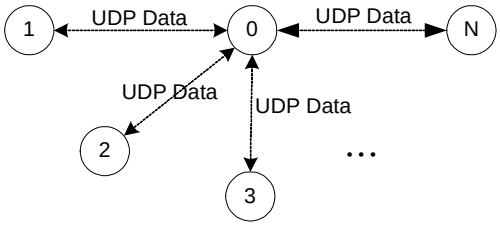


Figure 6.1: Star topology with multiple UDP flows

each destination in a table. Then, if the receiver overhears data transmission and detects that the current estimate is changed, it sends back channel information. In addition, if the receiver receives an RTS, it estimates the channel from the RTS and sends back channel information on the CTS.

To begin, we validate our design using UDP traffic, and then we present performance results for TCP traffic to study how metadata piggybacking has an effect on TCP performance for a variety of channels.

6.4.1 UDP Traffic

We designed these experiments to validate our design. We did this by measuring throughput with metadata piggybacking enabled and disabled as a function of the number of nodes for three different scenarios. We first consider a simple scenario having delayed ACKs or channel information. Then we consider a more complex scenario having both delayed ACKs and channel information. We expect that, for all the scenarios, the piggybacking enabled scheme will show an enhancement over the piggybacking disabled scheme and further that the improvement will become significant as the number of nodes increases. This is because the number of nodes corresponds to the traffic load of the metadata and thus the piggybacking enabled scheme saves more overhead as the number of nodes increases.

Figure 6.1 shows the topology used for our experiments. In this topology, node 0 is an access point and node 1 to N are the nodes that can communicate only

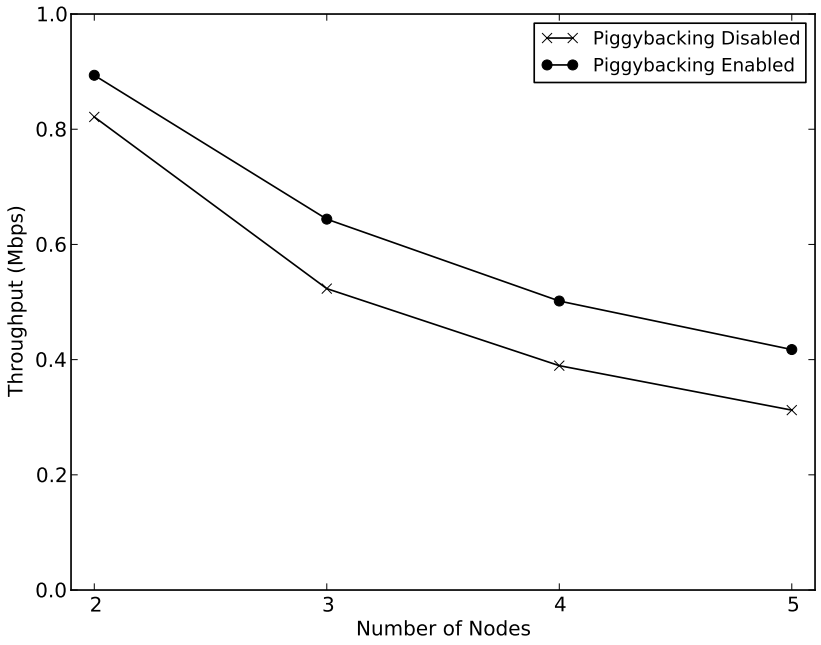


Figure 6.2: Throughput as a function of the number of nodes (delayed ACKs)

with the access point. For our experiments, the access point has N UDP flows, each of which travels towards a different receiver. In addition, each node has one UDP flow flowing toward the access point. All the UDP flows send packets with a fixed packet interval of 5 seconds and all nodes are within transmission range.

Delayed ACKs We first consider a simple scenario having delayed ACKs only. Figure 6.2 shows the average throughput for a single UDP flow as a function of the number of nodes. The X-axis is the number of nodes, and the Y-axis is the throughput in Mbps. For the X-axis, we did not count the access point. For this experiment, we used a small timeout value of 1 second for delayed ACKs. In addition, we will see how delayed ACKs have an impact on throughput, and thus we used rate adaptation using reciprocity, which does not require sending channel information.

Table 6.1: Performance analysis (delayed ACKs)

Number of Nodes	2	3	4	5
Frame Size (Piggybacking Enabled)	5.0KB	7.4KB	10.0KB	11.6KB
Frame Size (Piggybacking Disabled)	4.3KB	4.1KB	5.0KB	5.0KB
Total TXs (%)	78.3	53.8	46.2	39.1
Performance Gap (%)	8.5	23.1	28.2	35.5

The result shows that the performance gap between the piggybacking enabled and disabled schemes becomes more significant as the number of nodes increases. This is because, as the number of nodes increases, the access point (node 0) has more packets having different destinations, resulting in generating more delayed ACKs from the receivers. Thus, the piggybacking enabled scheme supports piggybacking of the delayed ACKs with user packets, which effectively reduces the overhead of sending the delayed ACKs.

Table 6.1 presents the performance analysis for delayed ACKs as a function of the number of nodes. The frame size indicates the average frame size for all nodes. In addition, the total TXs represents the number of transmissions for the piggybacking enabled scheme as a percentage of that for the piggybacking disabled scheme. The performance gap means the throughput gap between the piggybacking enabled and disabled schemes.

This analysis shows that, for the piggybacking enabled scheme, the average frame size increases significantly as the number of nodes increases. On the other hand, for the piggybacking disabled scheme, the increment of the average frame size is negligible. This is because delayed ACKs are very small, which has a significant impact on the average size. Thus, for a given file size, the piggybacking enabled scheme reduces the number of transmissions. The maximum throughput gap increases to 35.5% when the number of nodes is 5.

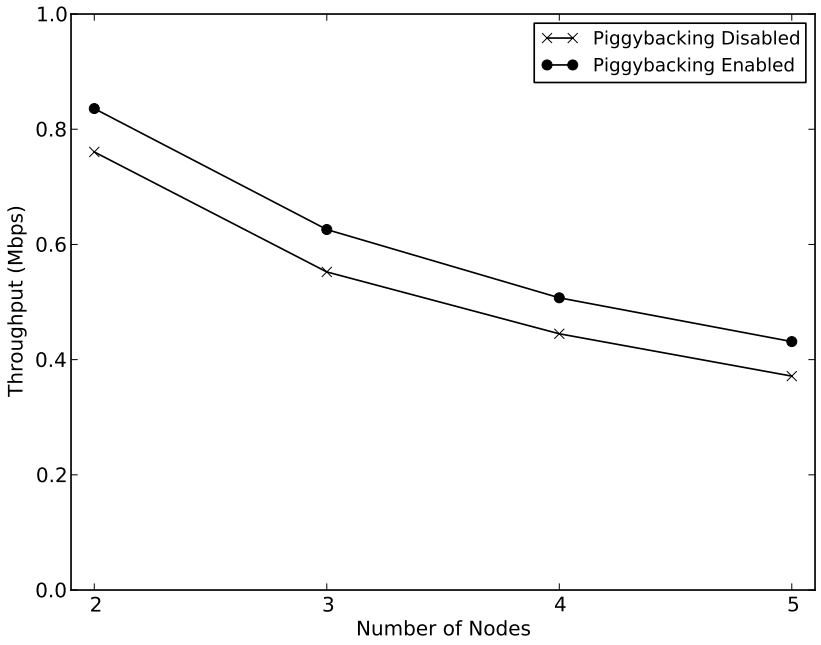


Figure 6.3: Throughput as a function of the number of nodes (channel information)

Channel information We consider another simple scenario having channel information only. Figure 6.3 shows the average throughput for a single UDP flow as a function of the number of nodes. The X-axis is the number of nodes, and the Y-axis is the throughput in Mbps. To see how channel information has an impact on throughput performance, we used rate adaptation using explicit feedback, which requires sending channel information for rate adaptation. In addition, we used the delayed ACK scheme using infinite amounts of delay, as described in Section 5.1.2, and thus we did not use timers for delayed ACKs.

The result shows that the piggybacking enabled scheme achieves some performance gain over the piggybacking disabled scheme. However, compared to the experimental result for delayed ACKs, the improvement becomes small even though channel information can be generated at all the nodes overhearing data transmis-

Table 6.2: Performance analysis (channel information)

Number of Nodes	2	3	4	5
Frame Size (Piggybacking Enabled)	4.6KB	7.1KB	8.6KB	12.1KB
Frame Size (Piggybacking Disabled)	3.6KB	4.5KB	6.1KB	6.9KB
Total TXs (%)	79.8	66.0	67.9	45.0
Performance Gap (%)	10.0	13.4	13.4	16.2

sions by neighboring nodes. This is because, in our environment, the channel is stable and our rate adaptation sends back channel information only when the channel changes. Thus, the performance gap entirely depends on how frequently the channel changes.

Table 6.2 presents the performance analysis for channel information as a function of the number of nodes. This analysis shows that, similar to the previous analysis, the gap of the average frame size between the piggybacking enabled and disabled schemes increases as the number of nodes increases. Thus, for a given file size, the piggybacking enabled scheme reduces the number of transmissions. The maximum throughput gap increases to 16.2% when the number of nodes is 5.

Delayed ACKs + Channel information We consider a complex scenario having both delayed ACKs and channel information. Figure 6.4 shows the average throughput for a single UDP flow as a function of the number of nodes. The X-axis is the number of nodes, and the Y-axis is the throughput in Mbps. The gray lines show the previous results for delayed ACKs or channel information for reference. To see how delayed ACKs and channel information have an impact on throughput performance, we used a small timeout value of 1 second for delayed ACKs and rate adaptation using explicit feedback.

The result shows that the piggybacking enabled scheme achieves significant improvement over the piggybacking disabled scheme. This is because, in this scenario, we have metadata of delayed ACKs and channel information, and the pig-

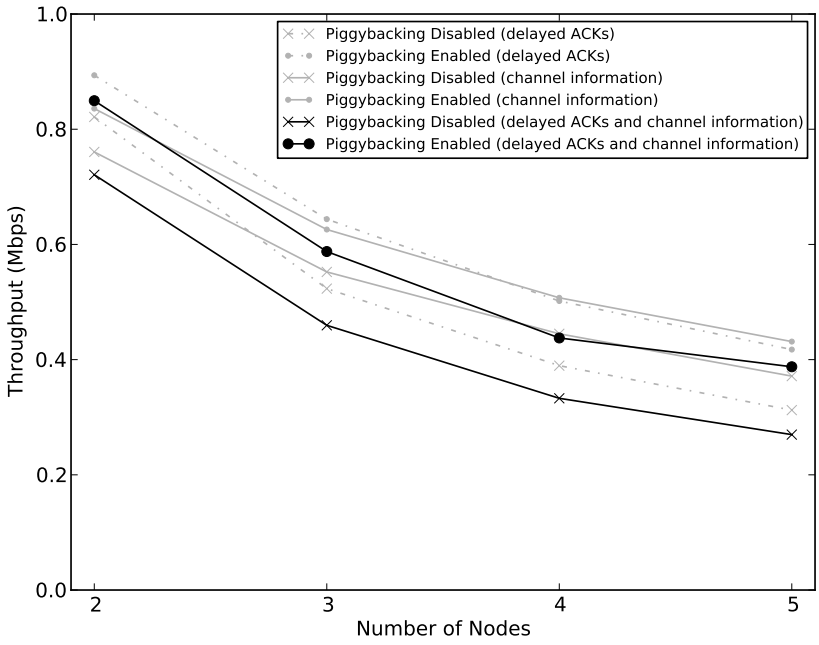


Figure 6.4: Throughput as a function of the number of nodes (delayed ACKs and channel information)

gybacking enabled scheme can save more overhead by piggybacking of the delayed ACKs and channel information with user packets.

Table 6.3 presents the performance analysis for delayed ACKs and channel information as a function of the number of nodes. This analysis shows that, for the piggybacking enabled scheme, the average frame size increases significantly as the number of nodes increases. On the other hand, for the piggybacking disabled scheme, the increment of the average frame size is negligible. This is because delayed ACKs and channel information are very small, which has a significant impact on the average size. Thus, for a given file size, the piggybacking enabled scheme reduces the number of transmissions dramatically. The maximum throughput gap increases to 44.4% when the number of nodes is 5.

Table 6.3: Performance analysis (delayed ACKs and channel information)

Number of Nodes	2	3	4	5
Frame Size (Piggybacking Enabled)	5.1KB	7.1KB	8.9KB	10.5KB
Frame Size (Piggybacking Disabled)	3.7KB	3.7KB	3.5KB	3.8KB
Total TXs (%)	68.9	45.9	34.4	27.1
Performance Gap (%)	18.0	28.3	33.3	44.4

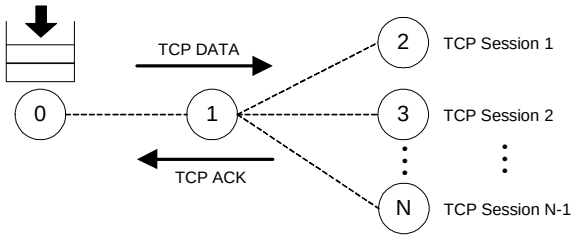


Figure 6.5: Star topology with multiple TCP flows

6.4.2 TCP Traffic

We designed these experiments to study how metadata piggybacking effects TCP performance for a variety of channels. We did this by measuring throughput for multi-destination aggregation schemes with no piggybacking, piggybacking of delayed ACKs, piggybacking of channel information, and piggybacking of both types of metadata as a function of the number of TCP sessions. We expect that the piggybacking enabled scheme will enhance throughput over the piggybacking disabled scheme and the amount of enhancement will increase as the number of TCP sessions increases. This is because the metadata piggybacking scheme can reduce overhead of sending metadata, and the amount of metadata increases as the number of TCP sessions increases.

Figure 6.5 shows the topology used for our experiment. This is the same topology as used in Chapter 5. In addition, we used a multi-destination aggregation scheme with rate adaptation using explicit feedback and a small timeout value of 1 second for delayed ACKs.

Static channel Figure 6.6 shows the average throughput for a single TCP flow as a function of the number of sessions for a static channel. The X-axis is the number of sessions, and the Y-axis is the throughput in Mbps. The result shows that multi-destination aggregation with piggybacking of delayed ACKs and channel information achieves a significant improvement in throughput over multi-destination aggregation with no piggybacking. In addition, we observe that the delayed ACKs have a greater impact on throughput than channel information when the number of sessions is 2 and 3. This is because we used a static channel that does not change very frequently, and our rate adaptation only sends channel information when the channel changes. However, the result shows that, as the number of sessions increases, channel information becomes more important than delayed ACKs. This is because channel information can be generated at all the nodes overhearing the data transmissions by neighboring nodes, which causes an exponential increase in the amount of channel information as the number of sessions increases.

Table 6.4 presents the performance analysis for multi-destination aggregation with no piggybacking and with piggybacking as a function of number of sessions for a real static channel. The frame size indicates the average frame size for all nodes. In addition, the total TXs represents the number of transmissions for multi-destination aggregation with piggybacking of both types of metadata as a percentage of that for multi-destination aggregation with no piggybacking. The performance gap means the throughput gap between multi-destination aggregation with piggybacking of both types of metadata and with no piggybacking.

This analysis shows that, for multi-destination aggregation with piggybacking of both types of metadata, the average frame size increases significantly as the number of sessions increases. On the other hand, for multi-destination aggregation with no piggybacking, the average frame size decreases slightly as the number of sessions increases. This is because, for multi-destination with no piggybacking, de-

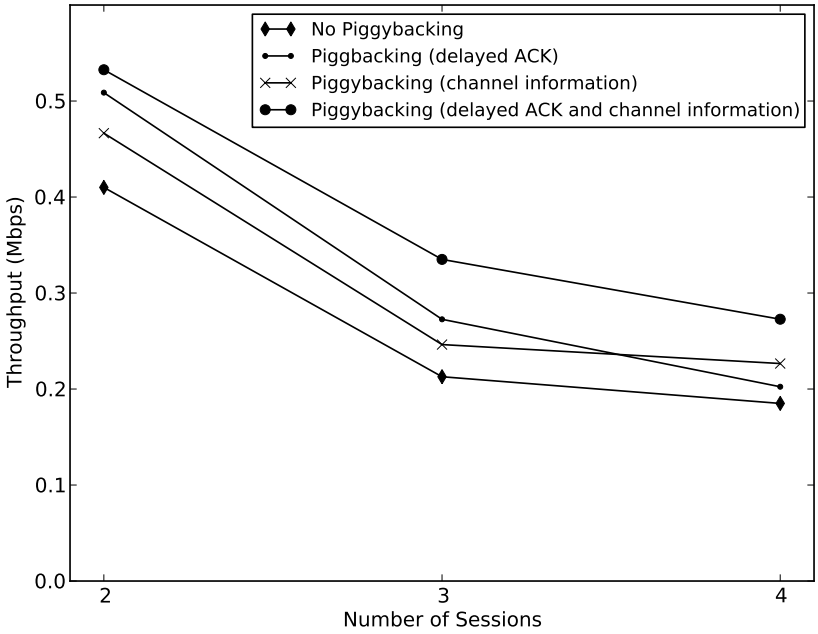


Figure 6.6: TCP throughput for a real static channel

layed ACKs and channel information are sent alone, which has a significant impact on the average size. Thus, for a given file size, multi-destination aggregation with piggybacking reduces the number of transmissions. The maximum throughput gap increases to 47.4% when the number of sessions is 4.

In addition, compared to multi-destination aggregation with no piggybacking, the increment of the average frame size for multi-destination aggregation with piggybacking of both types of metadata is always bigger than the sum of that for multi-destination aggregation with piggybacking of each of metadata. Further, the gap of the size increment increases as the number of sessions increases. This is because multi-destination aggregation with piggybacking of delayed ACKs or channel information still has metadata being sent alone, and thus each metadata transmission has an impact on other sessions' RTT, resulting in decrement of TCP trans-

Table 6.4: Performance analysis for a static channel

Number of Sessions	2	3	4
Frame Size (No Piggybacking)	2.0KB	1.9KB	1.9KB
Frame Size (Piggybacking of Delayed ACKs)	2.9KB	2.8KB	2.3KB
Frame Size (Piggybacking of Channel Information)	2.7KB	2.6KB	2.8KB
Frame Size (Piggybacking of Delayed ACKs and Channel Information)	3.7KB	3.9KB	5.8KB
Total TXs (%)	53.1	53.6	43.6
Performance Gap (%)	29.9	40.8	47.4

mission rates. On the other hand, multi-destination aggregation with piggybacking of both types of metadata greatly reduces the impact of sending metadata on TCP performance, resulting in more performance gain.

Time-varying channel We consider how metadata piggybacking effects TCP performance for a real time-varying channel. For this experiment, we are using a time-varying channel, and thus we expect that channel information will have a greater impact on network performance than delayed ACKs due to frequent channel changes.

Figure 6.7 shows the average throughput for a single TCP flow as a function of the number of TCP sessions for a real time-varying channel. The X-axis is the number of sessions, and the Y-axis is the throughput in Mbps. For this experiment, we created a real time-varying channel by oscillating a metal fan and mounting one antenna on each of the left and right sides of the fan at the central node (node 1 in Figure 6.5).

The result shows that multi-destination aggregation with piggybacking of delayed ACKs and channel information achieves a significant improvement in throughput over multi-destination aggregation with no piggybacking. In addition, we observe that multi-destination aggregation with piggybacking of channel information always outperforms multi-destination aggregation with piggybacking of delayed ACKs and further, as the number of sessions increases, the performance for multi-

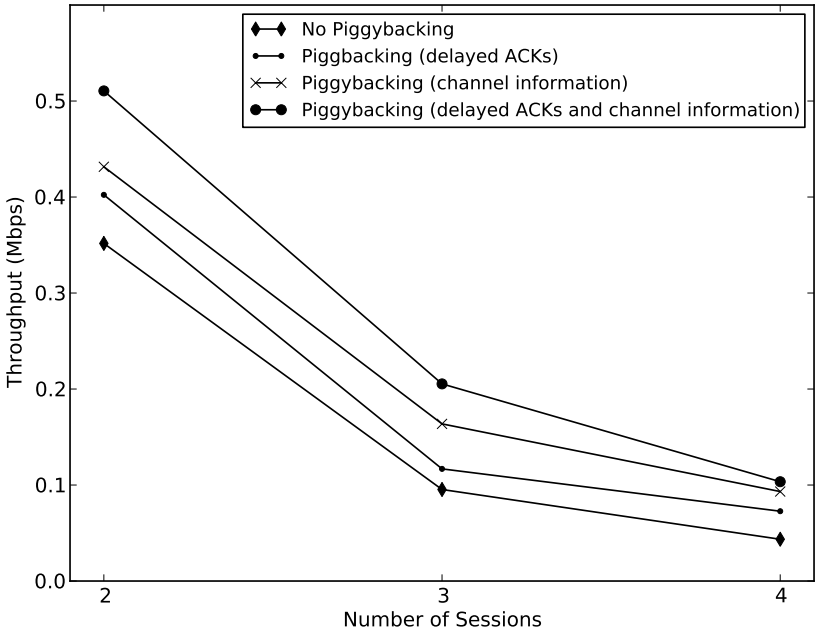


Figure 6.7: TCP throughput for a real time-varying channel

destination with piggybacking of channel information is close to that for multi-destination with piggybacking of channel information and delayed ACKs. This implies that, as expected, channel information has a greater impact on throughput than delayed ACKs.

Table 6.5 presents the performance analysis for multi-destination aggregation with no piggybacking and with piggybacking as a function of number of sessions. This analysis shows that, for multi-destination aggregation with no piggybacking, the average frame size decreases significantly as the number of sessions increases. On the other hand, for multi-destination aggregation with piggybacking of delayed ACKs and channel information, the average frame size is less sensitive to the number of sessions, resulting in the increased gap in the average frame size. Thus, for a given file size, multi-destination aggregation with piggybacking reduces the number

Table 6.5: Performance analysis for a time-varying channel

Number of Sessions	2	3	4
Frame Size (No Piggybacking)	1.99KB	0.95KB	0.66KB
Frame Size (Piggybacking of Delayed ACKs)	2.87KB	1.50KB	1.19KB
Frame Size (Piggybacking of Channel Information)	2.94KB	1.82KB	1.37KB
Frame Size (Piggybacking of Delayed ACKs and Channel Information)	3.63KB	2.42KB	2.67KB
Total TXs (%)	57.8	39.4	24.5
Performance Gap (%)	45.2	110.5	137.7

of transmissions. The maximum throughput gap increases to 137.7% when the number of sessions is 4.

In addition, compared to multi-destination with no piggybacking, the increment of the average size has a similar trend as that in the previous analysis. Thus, the increment for multi-destination aggregation with piggybacking of both types of metadata is similar to or bigger than the sum of that for multi-destination with piggybacking of each type of metadata. Further, as the number of sessions increases, the gap of the size increment increases, resulting in more performance gain for multi-destination aggregation with piggybacking of both types of metadata. However, when the number of sessions is 4, the performance gain is smaller than we expected. This is because, for multi-destination aggregation with piggybacking of both types of metadata, we observed that more retransmissions occur.

Chapter 7

Future Work

There are two main future directions for this work. One direction is to extend our design to provide further mechanisms that improve the flexibility and performance of our system. The second direction is to evaluate our design by performing a more extensive set of experiments. Here, we discuss the work we envision in each direction.

7.1 Design Extension

The first direction is to extend our design to provide further key mechanisms that support more flexibility and performance improvement of our system.

7.1.1 RTS/CTS Exchange

For our design, we assume that using a RTS/CTS exchange is always desirable. Using a RTS/CTS exchange has the advantage that we can prevent the hidden node problem. However, in the case where hidden nodes do not exist (e.g., all nodes are within transmission range), the cost of the RTS/CTS exchange is high. Thus, in this case, we might achieve a performance improvement by allowing ACK frames to convey rate and size information. However, a challenging issue is how to update the

most up-to-date rate when the channel changes in-between data transmissions. One possible solution is to use a timer. Thus, for data transmission, a sender uses the data rate obtained from the latest ACK when the sender transmits that data before a timer expires. If the timer expires, the sender can transmit a probe to update the data rate. Further, the timeout value can be chosen adaptively based on channel coherence time.

In addition to ACKs, we could introduce NACK (negative acknowledgement) frames. The advantages of using NACKs are: first, saving waiting time for timeout when a data frame is corrupted; and second, sending back some useful information for the corrupted data frame such as channel state information. Thus, we expect that using ACKs and NACKs will achieve some performance gain by saving the cost of RTS/CTS exchange.

7.1.2 Multi-Destination Aggregation

For multi-destination aggregation, our goal was to show that the idea of multi-destination aggregation is feasible and worth further exploration, not to find the best possible design. Our experimental results demonstrated the feasibility of our design and showed that multi-destination aggregation achieves a significant improvement in throughput and fairness over the no aggregation and single-destination aggregation schemes. As future work, we could optimize the multi-destination aggregation design, resulting in greater performance. One possible idea is to consider the quality of service for the multi-destination aggregation design. For our design, we aggregate multiple frames having different destinations on a first-come first-served basis. We could extend this design by giving a priority to each frame based on delay and jitter constraints and determining which frames need to be aggregated based on the priority. Another possible idea is to optimize block ACKs to improve efficiency in the multi-destination aggregation design. For our design, we used block ACKs

that allow sending an individual ACK for each subframe by using a bitmap structure, which follows the 802.11n standard. The block ACKs always have a fixed size bitmap information regardless of how many subframes have to be acknowledged. This results in inefficient bandwidth consumption in the case when the number of subframes to be acknowledged is less than the bitmap size. Thus, we could improve flexibility in block ACKs by adapting the bitmap size based on how many subframes need to be acknowledged.

In addition, we could optimize the aggregation format, described in Appendix C.1, to achieve more performance gain. Our multi-destination aggregation format allows a fixed number of destinations to be aggregated in a single transmission. However, we could allow aggregating arbitrary numbers of different destinations by modifying the PHY header described in Appendix C.1. One possible solution is to put rate and length information for each destination right before the unicast portion having that destination. We expect that this will improve aggregation efficiency by aggregating unicast frames having more destinations into a single transmission.

7.1.3 Metadata Piggybacking

Our approach allows piggybacking of a variety of metadata with user packets on a first-come first-served basis. However, in some cases, it might be important to give a higher priority to metadata for some destinations. As an example, suppose that a node has more queueing than some other nodes. In this case, a delayed ACK for that node needs to be delivered earlier than another node by giving a higher priority to that node. We expect that this will make our approach more flexible, resulting in performance improvement.

We could consider sharing each node's queueing information with neighboring nodes. Choi [45] proposed a scheme that controls the contention window by

exchanging queueing information, the goal of which is to improve throughput by mitigating unfairness between nodes. To make this coordination effective, queueing information needs to be shared with all neighbors, and thus we categorize this as broadcast. Thus, we expect that piggybacking queueing information with user packets will reduce the overhead of sending this information, which results in more improvement.

7.2 Evaluation of Detailed Mechanisms

The second direction for future work is to further evaluate the detailed mechanisms of our design by performing more experiments and analysis.

First, we could extend our experimental results for the rate-adaptive frame aggregation scheme with rate adaptation switching between single stream and double stream rates. We expect even more advantages in this case, since we will be able to double our maximum possible rate. Thus, compared to single stream rates, the number of bits that can be aggregated into a single transmission for a given time-varying channel will be doubled. This is because the double stream rates increase the number of bits that a single sample carries by twice by transmitting two different streams on two antennas.

Second, for our experiments, we used a static routing protocol to force the topologies we are interested in. This is because all of our nodes are within the transmission range of the others, and thus they would have not discovered any multi-hop routes. However, to study how ad-hoc routing protocols impact frame aggregation, we could perform experiments with some ad-hoc routing protocols such as AODV (ad-hoc on-demand distance vector) and DSR (dynamic source routing) [32, 33]. Here, a challenging experimental issue is to control each node's transmission power to create multi-hop paths. We expect that, with ad-hoc routing protocols, our techniques will achieve more performance gain. This is because, unlike the static

routing protocol, ad-hoc routing protocols involve sending flooding packets to maintain multi-hop routes, and our experimental results showed that our techniques greatly reduce the impact of flooding on network performance. In addition, ad-hoc routing protocols involve sending metadata for neighbor discovery, and an issue is to maximize service availability with minimizing bandwidth consumption [46]. Our techniques provide a way to reduce overhead of sending metadata by piggybacking of metadata with user packets, resulting in more improvement. Thus, we expect that our techniques could achieve a significant improvement in throughput by minimizing the impact of metadata and flooding on network performance.

Finally, to validate our design, we performed all the experiments in a small lab space. In this environment, it was difficult to create a variety of real channels. To create real channels that vary significantly in the time and spatial domains, our best solution involved a metal fan whose head oscillates back and forth with one antenna mounted on each of the left and right sides of the fan. Thus, it would be interesting to perform some experiments in outdoor environment. This is because the outdoor environment will have different channel properties from indoor environment, and it is an open space, where we would see more complex channels.

Chapter 8

Contributions and Conclusions

Our main contribution is to demonstrate that we are able to improve network performance by designing and implementing new frame aggregation techniques. These techniques improve aggregation efficiency by taking advantage of rate adaptation and the broadcast nature of wireless transmissions. Further, we validated our designs by implementing them using our wireless node prototype, Hydra, conducting extensive experiments and measuring system performance. Detailed contributions are as follows:

- **Fixed-Rate Frame Aggregation:** We developed a system that can aggregate both unicast and broadcast frames. Further, the system can classify TCP ACK segments so that they can be aggregated with TCP data flowing in the opposite direction. Contributions of this work are:
  - Creation of a framework to support aggregating broadcast frames and unicast frames having one destination.
  - Development of a cross-layer design that can aggregate TCP data and TCP ACKs flowing in the opposite direction by treating TCP ACKs as broadcasts.

- Implementation of our design using our wireless node prototype, Hydra.
- Validation of our design by conducting extensive experiments on real channels.
- Achievement of throughput improvement over the no aggregation and single-destination aggregation schemes.
- Demonstration that aggregation envisioned by 802.11n frame aggregation is a good idea in a concrete way.
- Demonstration that broadcast aggregation, taking advantage of the broadcast nature of wireless transmissions, improves aggregation efficiency, resulting in more performance gain.
- Demonstration that our cross-layer approach, classifying TCP ACKs as broadcasts, improves TCP performance by taking advantage of the broadcast nature of wireless transmissions.
- Demonstration that TCP effectively expands its window to take advantage of aggregation, resulting in the queueing needed for frame aggregation to work effectively.

- **Rate-Adaptive Frame Aggregation:** We developed a rate-adaptive frame aggregation scheme that combines broadcast aggregation with rate adaptation. Contributions of this work are:
  - Pre-design experiments and analysis to study the impact of aggregation size on performance in time-invariant and time varying channels.
  - Pre-design experiments and analysis to study the impact of block ACKs on rate adaptation and frame aggregation for time-invariant and frequency selective channels.
  - Design of rate-adaptive frame aggregation that adapts the best aggregation size based on channel and data rate chosen by rate adaptation.

- Implementation of our design using our wireless node prototype, Hydra.
- Validation of our design by conducting extensive experiments on real and emulator-based channels.
- Achievement of a significant improvement in throughput over the no aggregation and fixed-size aggregation schemes.
- Demonstration that rate adaptation has an important role to play in the performance of frame aggregation.
- Demonstration that using block ACKs reduces the design cost by decoupling rate adaptation from frame aggregation.
- Demonstration that, for a given channel and data rate, there is a best aggregation size, maximizing throughput, and thus it is important to adapt aggregation size based on the data rate and channel.

• **Multi-Destination Frame Aggregation:** We designed a multi-destination frame aggregation scheme to aggregate broadcast frames and unicast frames that are destined for different receivers. Contributions of this work are:

- Extension of the aggregation framework to support aggregating broadcast frames and unicast frames having different destinations.
- Development of three key mechanisms to support multi-destination aggregation: controlling link-level ACKs, supporting multiple rates, and supporting size adaptation.
- Implementation of our design using our wireless node prototype, Hydra.
- Validation of our design by conducting extensive experiments on real and emulator-based channels.
- Achievement of a significant improvement in throughput and fairness over the no aggregation and single-destination aggregation schemes.

- Demonstration that our design, supporting multi-destination aggregation in a general way, achieves the same advantages that the special-purpose design had for TCP ACKs.

- **Metadata Piggybacking:** We extended multi-destination rate-adaptive frame aggregation to allow piggybacking of various types of metadata with user packets. Contributions of this work are:

  - Development of a design to support piggybacking of various types of metadata with user packets.
  - Implementation of our design using our wireless node prototype, Hydra.
  - Validation of our design by conducting extensive experiments on real channels.
  - Achievement of a significant improvement in throughput over the piggybacking disabled scheme.
  - Demonstration that our design, piggybacking of metadata onto an existing transmission, lowers the impact of metadata-based control protocols on data transport.

8.1 Conclusions

Transmissions in a wireless network incur significant overhead for PHY headers and acquiring the floor. These overheads are not reduced by using higher data rates and can dominate performance at high rate, negating the advantages of high performance PHYs and rate adaptation. Frame aggregation, where more than one frame is aggregated into a single transmission, addresses this problem by amortizing these overheads over more data and reducing the total number of transmissions. However, traditional aggregation schemes require that frames that are aggregated all be

destined to the same receiver. These approaches neglect the fact that transmissions are broadcast, and a single transmission will potentially be received by many nodes.

To address these problems, we developed new aggregation techniques that take advantage of rate adaptation and the broadcast nature of wireless transmissions. We first designed a frame aggregation scheme that can aggregate both unicast and broadcast frames. Further, we treated TCP ACKs as broadcasts so as to enable them to be aggregated with TCP data flowing in the opposite direction. Second, we developed a rate-adaptive frame aggregation scheme that allows us to find the best aggregation size by tracking the size based on received data frames and the data rate chosen by rate adaptation. Third, we developed a multi-destination frame aggregation scheme to aggregate broadcast frames and unicast frames that are destined for different receivers using delayed ACKs. Finally, we extended multi-destination rate-adaptive frame aggregation to support piggybacking of various types of metadata with user packets.

We implemented and validated our design not through simulation, but rather using our wireless node prototype, Hydra, which supports a high performance PHY based on 802.11n. To validate our design, we conducted extensive experiments both on real and emulator-based channels and measured system performance. The experimental results demonstrated the feasibility of our design and showed that our design can achieve significant performance improvement, especially at a high rate.

One important lesson we learned from this study is to gain insight into how our aggregation techniques have an effect on network performance. We showed four different aggregation techniques and studied the impact of our techniques on network performance throughout extensive experiments and analysis. First, it is important for frame aggregation to take advantage of the broadcast nature of wireless transmissions. Our aggregation techniques leverage the broadcast nature of wireless transmissions by allowing subframes to be sent to more than one destination. This

increases the level of aggregation, resulting in reducing overhead. In addition, this reduces the total number of transmissions, resulting in less time waiting for the floor and transmitting control frames. Second, it is important to maximize synergy between frame aggregation and rate adaptation. Experimental results showed that our rate-adaptive frame aggregation, adapting the aggregation size based on data rate chosen by rate adaptation, achieves a significant improvement in throughput for time-varying channels. Third, TCP has an important role to play in the performance of frame aggregation. TCP effectively expands its window to take advantage of aggregation, resulting in enough queueing to support frame aggregation. Further, our aggregation techniques improve the TCP capacity significantly and allow fast increment of congestion window size until reaching the capacity. Experimental results showed that our aggregation techniques achieve a significant improvement in TCP performance. Thus, we expect that our design can be a promising solution to improve network performance, especially with TCP and high performance PHYs.

Appendix A

Implementation of Fixed-Rate Frame Aggregation

We implemented this protocol using Hydra, our wireless node/network prototype. We enhanced the Hydra MAC and PHY to support unicast, broadcast, and TCP ACK aggregation as described in Section 3.2. We describe the MAC subframe format used by our aggregation schemes, the receive and transmit processes, and finally details of the classification of TCP ACKs as broadcasts.

A.1 Frame

A single physical frame includes a series of MAC subframes. These subframes are embedded in the aggregated frame shown in Figure 3.2. Figure A.1 shows the format of each MAC subframe. This follows the standard 802.11 MAC format with the exception that we eliminated the address 4 field because we do not support infrastructure networking. Each subframe includes a MAC header containing general information: duration, source and destination addresses, and length. Our aggregation protocol only uses the duration field of the first unicast subframe for virtual

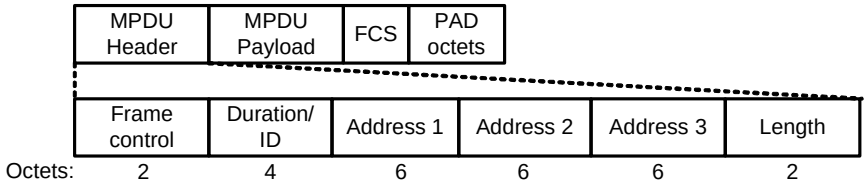


Figure A.1: MAC subframe format

carrier sensing. However, for the purpose of easy prototyping, all of the subframes have the duration field. Each subframe includes a 2-byte length field. Finally all of the subframes contain frame check sequence (FCS) and PAD octets.

Frames transmitted in the broadcast portion of the frame can have a broadcast or unicast address but are not acknowledged. On the other hand, the subframes transmitted in the unicast portion require an ACK and thus in this design all must be addressed to the same destination.

A.2 The Receive Process

When receiving a frame, the PHY uses the broadcast length and rate information to decode the broadcast subframes and then the unicast length and rate information to decode the unicast subframes. Once the PHY completes decoding all the subframes, it sends the subframes up to the MAC. When the MAC receives an aggregated frame, it first processes the broadcast subframes and then processes the unicast subframes. For the broadcast portion, as soon as each subframe passes the CRC, the MAC sends the subframe to the next layer. Thus the broadcast subframes do not suffer from higher loss probability though they are aggregated with unicast subframes. For the unicast subframes, the MAC checks the destination address and all of the CRCs, and, if they all pass, then the MAC sends them up to the next layer and sends a link-level ACK. Otherwise, all of the unicast subframes are discarded. We could optimize by storing and applying CRCs to aggregates instead of individual

subframes. However, the current scheme has only a small overhead and our design, discussed in Chapter 4, extends the current design to a block ACK scheme like that of 802.11n.

A.3 The Transmit Process

On the transmit side, the MAC must assemble the aggregated frames into the correct format. To achieve this, we have two queues: one for broadcasts and one for unicasts. The MAC first searches the broadcast queue and assembles all the broadcast frames. Then the MAC searches the unicast queue and gathers the unicast frames being transmitted to the same destination as the first frame in the unicast queue. This breaks queueing semantics because this might change the service order of incoming packets. We further discuss this issue in Appendix B. Once completed, the MAC aggregates the broadcast subframes followed by unicast subframes up to a parameterized maximum aggregation size. Putting the broadcasts ahead of the unicasts enables the broadcasts to be less sensitive to changes in the wireless channel. This is because the channel might change during transmission and the subframes close to the PHY training sequences are less likely to be corrupted by these changes. Once the frames are assembled, the MAC hands the aggregated frame down to the PHY along with the rate and length information for the broadcast and unicast parts of the frame. The entire transmit process triggers when the DCF of the MAC acquires the floor.

A.4 TCP ACKs

The process above neglects TCP ACKs, which are specially handled when assigning packets to the unicast or broadcast queues. We assign "pure" TCP ACKs to the broadcast queue. TCP ACK segments that contain any data or are part of connec-

tion set-up cannot be treated in this way. This is because these are required to be reliable. All others are classified as pure TCP ACKs. Click [21] provides a packet classification mechanism, and our implementation uses these classifiers to sort pure TCP ACKs from other unicast frames and place them in the broadcast queue.

Appendix B

Implementation of Rate-Adaptive Aggregation

We implemented this protocol using Hydra. Basically our implementation extends broadcast aggregation, described in Chapter 3, to support our rate-adaptive frame aggregation. In the previous design, we used two queues: one for broadcast and one for unicast. However, this might change the service order in-between incoming broadcast and unicast packets, which changes the basic semantics. Thus, for this implementation, we use a single queue for all packets rather than individual broadcast and unicast queues.

To begin, we describe the MAC subframe format used by our rate-adaptive frame aggregation scheme and the RTS/CTS exchange that is required for the size and rate adaptation. Then we describe the receive process. Finally we describe our implementation of virtual carrier sensing.

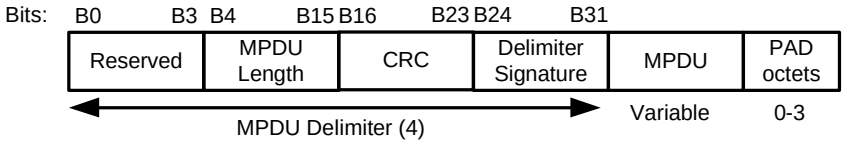


Figure B.1: Aggregated MAC subframe format

B.1 Frame

Figure B.1 shows the format of each subframe which follows the IEEE 802.11n A-MPDU format [4]. A single physical frame includes a series of MAC subframes. Each subframe contains a delimiter, a MAC protocol data unit (MPDU), and PAD octets. The MPDU includes a MAC header, payload, and 4-byte CRC. The delimiter contains a length, 1-byte of CRC and delimiter signature. The 1-byte CRC is used to detect errors for the length and the delimiter signature is used to detect the starting point of the subframe. Each MPDU includes a 4-byte CRC which enables the MAC to detect errors for individual subframes. The individual ACK for each subframe is delivered by the block ACK. Finally the PAD octets make each subframe to be a multiple of 4 octets in length.

B.2 RTS/CTS Exchange

Our design uses the RTS/CTS exchange for rate and size adaptation. Before a data transmission, a sender transmits a RTS and a receiver responds to the RTS by sending a CTS with the measured SNR and size determined by the adaptation control module. We return SNR rather than rate to support multi-destination frame aggregation described in Chapter 5.

Figure B.2 presents a block diagram of this exchange. Whenever a transmitter has data to send, it first transmits an RTS that includes a bound on the largest possible aggregate based on what packets are actually queued and available

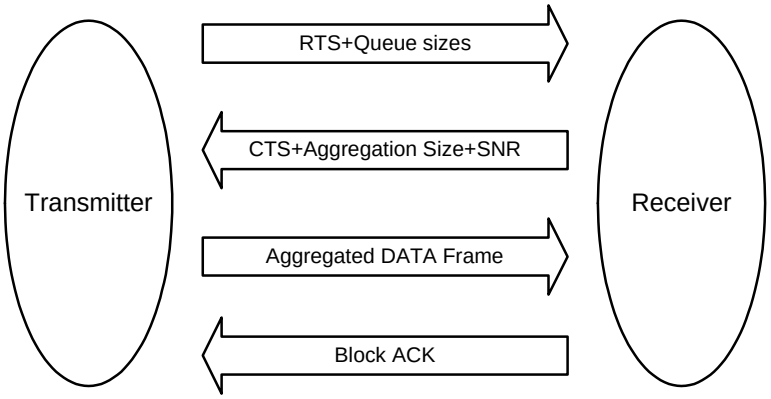


Figure B.2: Block diagram for RTS/CTS exchange

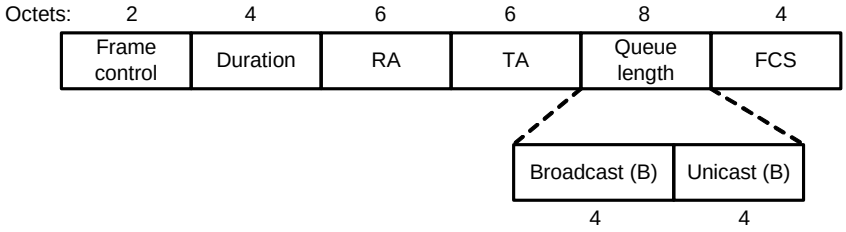


Figure B.3: Modified RTS frame format

to be aggregated. This bound is needed so that the CTS can update the NAV correctly even if the transmitter does not have enough data to fill the maximum aggregation size supported by the channel. From the RTS, a receiver first chooses the rate based on the measured SNR, and then selects the aggregation size for that rate. This size is determined by the minimum of the queue size from the RTS and the best aggregation size from the adaptive control module. The SNR and size are sent to the transmitter in the CTS. After receiving the CTS, the transmitter sends an aggregated frame based on the size and rate chosen by the SNR. Finally the receiver responds to the data frame by sending a block ACK indicating pass or fail for individual subframes. At this time, the adaptation control module updates the aggregation size (in samples) based on the received data frame.

Supporting rate and size adaptation requires modest changes to the RTS,

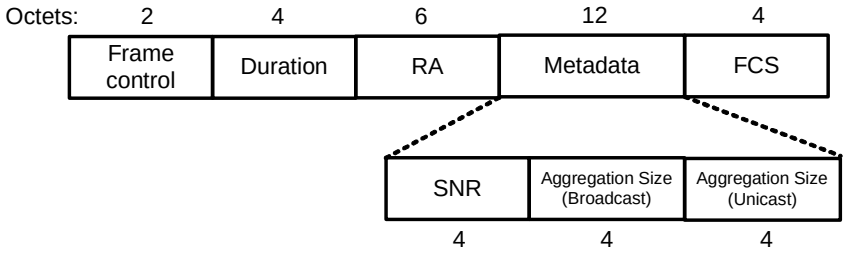


Figure B.4: Modified CTS frame format

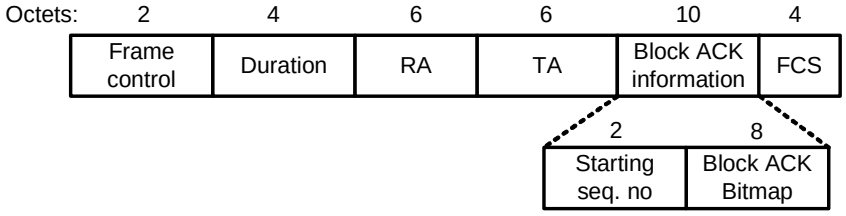


Figure B.5: Block ACK format

CTS, and ACK frames as shown in Figure B.3, B.4, and B.5 respectively. The RTS is modified by piggybacking queue lengths for broadcast and unicast frames. The modified CTS includes 12 bytes of metadata which contains SNR and aggregation sizes for broadcast and unicast frames. The ACK enables the MAC to send individual ACK for each subframe by using a bitmap structure, which follows the IEEE 802.11n standard [4].

B.3 The Receive Process

The receive process follows the IEEE 802.11n MAC [4] to deaggregate a frame. Further, the adaptation control module, residing at the receiver, tracks the best aggregation size using information from the deaggregation process.

When the MAC receives an aggregated frame, it first searches for a MAC delimiter and, if found, checks the 1-byte CRC for a length field of each subframe. If it passes, the MAC checks the 4-byte CRC for data portion of the subframe. If all

pass, the MAC sends the subframe to the next layer. Otherwise, the MAC drops the subframe and searches for the next subframe. After processing all the subframes, the MAC constructs a block ACK containing pass or fail for individual subframes, and then sends this to the transmitter. This allows unicast aggregation only and our implementation, discussed in Appendix C, extends the current implementation to support broadcast aggregation.

From the deaggregation process, the adaptation control module acquires the data rate which the frame was sent with, the number of successful subframes, and the status of the frame reception (i.e., success or burst drop). All the information is used by the module to update the success count and the best aggregation size in samples.

B.4 Virtual Carrier Sensing

When a transmitter has data to send, it first sends a RTS with a long NAV [4] calculated by the maximum A-MPDU size and the base rate. A receiver responds to the RTS by sending a CTS with the exact duration, which is calculated by the rates based on the measured SNR and the aggregation size. At this time, all the neighbors overhearing the CTS updates their own NAV. When the transmitter sends an aggregated frame with the exact duration, then all the neighbors overhearing the frame can update their own NAV. Finally, as a response to the aggregated frame, the ACK clears the NAV by setting the duration field to 0.

Appendix C

Implementation of Multi-Destination Aggregation

We implemented this protocol using Hydra. We enhanced rate-adaptive aggregation to support multi-destination aggregation. To begin, we describe the aggregation format used by multi-destination aggregation. Then, we describe the details of the receive and transmit processes.

C.1 Aggregation Format

Figure C.1 shows the aggregation format to support multi-destination aggregation, where B stands for broadcast portion and U_N stands for N-th unicast portion. Thus, in an aggregate, the broadcast portion is located at the front, followed by a series of unicast portions, each of which can include multiple subframes having

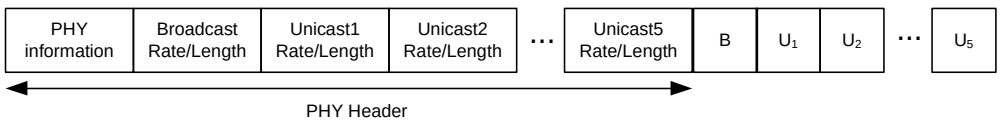


Figure C.1: Multi-destination aggregation format

the same destination. The location of each unicast portion for one destination is determined by where the first unicast frame having that destination is placed in the queue. Thus, the unicast portion for the destination of the data at the head of queue precedes the unicast portions for other destinations than head of queue. Putting the broadcasts ahead of the unicasts enables the broadcasts to be less sensitive to changes in the wireless channel. This is because the channel might change during transmission and the subframes close to the PHY training sequences are less likely to be corrupted by these changes. We do this because broadcast frames do not require link-level ACKs and thus sending broadcast frames is less reliable than sending unicast frames. We modified the PHY header format to support up to 6 different data rates in an aggregate. In addition, the format of each subframe follows the IEEE 802.11n A-MPDU format [4], as shown in Figure B.1.

C.2 Receive Process

The receive process enhances the IEEE 802.11n MAC [4] to deaggregate both broadcast and unicast subframes. Further, the adaptation control module, residing at the receiver, tracks the best aggregation size using the information from the deaggregation process.

When receiving a frame, the PHY uses the broadcast length and rate information to decode the broadcast subframes and then a series of unicast length and rate information to decode the unicast subframes in each unicast portion. Once the PHY completes decoding all the subframes, it sends the subframes up to the MAC. When the MAC receives an aggregated frame, it first processes the broadcast subframes and then processes the unicast subframes. For the subframes both in broadcast and unicast portions, the MAC first searches for a MAC delimiter and, if found, checks the 1-byte CRC for a length field of each subframe. If it passes, the MAC checks the 4-byte CRC for data portion of the subframe.

For the broadcast portion, as soon as each subframe passes all the CRCs, the MAC sends the subframe to the next layer. For each unicast portion, the MAC checks the destination address and all the CRCs. If all pass, the MAC sends the subframe to the next layer. Otherwise, the MAC drops the subframe and searches for the next subframe. After processing all the subframes, the MAC constructs a block ACK containing pass or fail for individual subframes. If the MAC is the destination for the first unicast portion, it sends back an ACK to the transmitter immediately. Otherwise, the MAC delays sending the ACK, as discussed in Section 5.1.2.

From the deaggregation process, the adaptation control module acquires the data rate which the frame was sent at, the number of successful subframes, and the status of the frame reception (i.e., success or burst drop). All the information is used by the module to update the success count and the best aggregation size in samples. Note that, to calculate the best size, the module considers broadcast subframes, unicast subframes having its own destination, and unicast subframes having other destinations using the same or lower rates than the unicast subframe having its own destination.

C.3 Transmit Process

On the transmit side, the MAC assembles the aggregated frames from a single queue for broadcast and unicast frames. Aggregating multiple frames depends on the type of data at the head of queue. If the head of queue is a unicast, the MAC first gathers the unicast frames being transmitted to the same destination as the head of queue. Then, the MAC searches for broadcast frames in the queue, and, if found, the MAC gathers a fixed number of broadcast frames. Then, if we still have space, the MAC gathers unicast frames having different destination from the head of queue. On the other hand, if the head of queue is a broadcast, the MAC searches for the first unicast frame in the queue to control the aggregate. Then, the MAC

gets the best aggregation size for the destination of the unicast frame based on a RTS/CTS exchange. Then, the MAC gathers a fixed number of broadcast frames. Then, the MAC gathers the first unicast frame and all unicast frames having the same destination as the first unicast up to the best size. Then, if we still have space, the MAC gathers the next unicast frame having different destination from the first unicast and all unicasts having that destination. The MAC can iterate this process up to 5 different destinations up to the best size. This is because the PHY header can maximally include 5 different unicast rate/length fields.

Once completed, the MAC aggregates the broadcast and unicast subframes into the correct format. Once the subframes are assembled, the MAC hands the aggregate down to the PHY along with the rate and length information for the broadcast and unicast portions. The entire transmit process triggers when the DCF of the MAC acquires the floor.

Appendix D

Implementation of Metadata Piggybacking

We implemented this protocol using Hydra. Basically our implementation extends multi-destination aggregation, described in Chapter 5, to support metadata piggybacking. For metadata piggybacking, we used the same aggregation format described in Appendix C.1. Here, we describe the details of the transmit process.

D.1 Transmit Process

When delayed ACKs or channel information needs to be sent, the MAC first constructs a unicast frame based on the destination address. As shown in Figure D.1

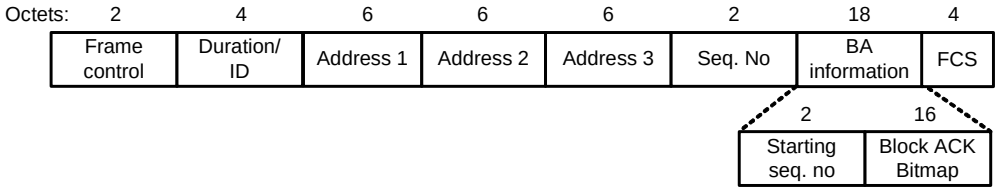


Figure D.1: Metadata frame format (delayed ACK)

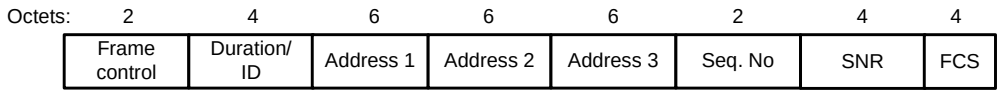


Figure D.2: Metadata frame format (channel information)

Table D.1: Frame control field for metadata

	Type (2 bits)	Subtype (4 bits)
Delayed ACK	10	1000
Channel Information	10	1001

and D.2, this frame can include the payload of bitmap information for the delayed ACK or SNR for the channel information. To decode this frame correctly, we define a new subframe type for each metadata by using the reserved values in the 802.11 MAC frame format [1]. Table D.1 shows the subframe type for each metadata.

For piggybacking of metadata with user frames, the MAC first aggregates all the frames having the same destination as the frame at the head of queue. Then, if we still have space, the MAC searches the queue to find whether there is metadata having that destination. If found, the metadata is aggregated with those frames. Note that an aggregate can include both channel information and delayed ACKs with user frames. Then, if we still have space, the MAC does the same process for the frames having different destinations up to 5 different unicast destinations.

Further, if the channel is static and thus the MAC uses 2% overhead bound for an aggregate, the MAC can place metadata beyond the 2% overhead bound to aggregate the metadata with user frames.

Bibliography

[1] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, Part 11 standard ed., IEEE 802.11 Working Group, 1999.

[2] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: High-Speed Physical Layer in the 5 GHz Band*, Part 11 standard ed., IEEE 802.11 Working Group, Sep 1999.

[3] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification: Further Higher-Speed Physical Layer Extension in the 2.4 GHz Band*, IEEE 802.11 Working Group, 2003.

[4] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification-Draft 5.0: Enhancements for Higher Throughput*, Part 11 standard ed., IEEE 802.11n Working Group, 2008.

[5] "Hydra – A Wireless Multihop Testbed," http://hydra.ece.utexas.edu/.

[6] L. L. Peterson and B. S. Davie, *Computer Networks: A Systems Approach*, 3rd ed. Morgan Kaufmann, 2003.

[7] R. van Renesse, "Masking the overhead of protocol layering," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 4, pp. 96–104, Oct. 1996.

[8] J. Moon and H. Y. Yeom, "IP Concatenation: The Method for Enhancement

of IPsec Performance," *Lecture Notes in Computer Science*, vol. 2720/2003, pp. 365–374, Mar. 2003.

[9] R. Raghavendra, A. P. Jardosh, E. M. Belding-Royer, and H. Zheng, "IPAC: An IP-based Adaptive Packet Concatenation for Multihop Wireless Networks," in *Proceedings of IEEE ACSSC*, Nov. 2006.

[10] *HT MAC Specification v1.24*, Enhanced Wireless Consortium, 2006.

[11] Y. Kim, S. Choi, K. Jang, and H. Hwang, "Throughput Enhancement of IEEE 802.11 WLAN via Frame Aggregation," in *Proceedings of IEEE VTC*, Sept. 2004.

[12] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly, "Opportunistic Media Access for Multirate Ad hoc Networks," in *Proceedings of ACM MOBICOM*, Sept. 2002.

[13] D. Skordoulis, Q. Ni, H.-H. Chen, A. P. Stephens, C. Liu, and A. Jamalipour, "IEEE 802.11n MAC Frame Aggregation Mechanisms for Next-Generation High-Throughput WLANs," *IEEE Wireless Communications*, vol. 15, no. 1, pp. 40–47, Feb 2008.

[14] S. Kim, S. Choi, Y. Kim, and K. Jang, "MCCA: A High-Throughput MAC Strategy for Next-Generation WLANs," *IEEE Wireless Communications*, vol. 15, no. 1, pp. 32–39, Feb 2008.

[15] W. Kim, H. K. Wright, and S. M. Nettles, "Improving the Performance of Multi-hop Wireless Networks using Frame Aggregation and Broadcast for TCP ACKs," in *Proceedings of ACM CoNEXT*, Dec. 2008.

[16] W. Kim, M. O. Khan, K. T. Truong, S.-H. Choi, R. Grant, H. K. Wright, K. Mandke, R. C. Daniels, R. W. Heath, Jr., and S. M. Nettles, "An Experimen-

tal Evaluation of Rate Adaptation for Multi-Antenna Systems," in *Proceedings of IEEE INFOCOM*, Apr. 2009.

[17] K. Mandke, S.-H. Choi, G. Kim, R. Grant, R. C. Daniels, W. Kim, S. M. Nettles, and R. W. Heath, Jr., "Early Results on Hydra: A Flexible MAC/PHY Multihop Testbed," in *Proceedings of IEEE VTC*, Apr. 2007.

[18] K. Mandke, R. C. Daniels, S.-H. Choi, S. M. Nettles, and R. W. Heath, Jr, "Physical Concerns for Cross-Layer Prototyping and Wireless Network Experimentation," in *Proceedings of ACM WiNTECH*, Sept. 2007.

[19] "GNU Radio: Universal Software Radio Peripheral," http://www.gnuradio.org/trac/wiki/USRP.

[20] "GNU Software Radio," http://www.gnu.org/software/gnuradio/.

[21] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[22] G. Holland, N. H. Vaidya, and P. Bahl, "A Rate-Adaptive MAC protocol for Multi-Hop Wireless Networks," in *Proceedings of ACM MOBICOM*, July 2001.

[23] A. Kamerman and L. Monteban, "WaveLAN II: A High-Performance Wireless LAN for the Unlicensed Band," *Bell Labs Technical Journal*, vol. 2, no. 3, pp. 118–133, Aug. 2002.

[24] R. W. Heath, Jr. and A. J. Paulraj, "Switching Between Diversity and Multiplexing in MIMO Systems," *IEEE Transactions on Communications*, vol. 53, no. 6, pp. 962–968, June 2005.

[25] W. C. Jakes, *Microwave Mobile Communications*. Wiley, 1974.

[26] A. Goldsmith, *Wireless Communications*. Cambridge University Press, 2005.

[27] A. Forenza, M. Airy, M. Kountouris, R. W. Heath, Jr., D. Gesbert, and S. Shakkottai, "Performance of the MIMO Downlink Channel with Multi-Mode Adaptation and Scheduling," in *Proceedings of IEEE SPAWC*, June 2005.

[28] C. Parsa and J. J. Garcia-Luna-Aceves, "Improving TCP Performance over Wireless Networks at the Link Layer," *Mobile Networks and Applications*, vol. 5, no. 1, pp. 57–71, Mar. 2000.

[29] J. Tourrilhes, "PiggyData: Reducing CSMA/CA Collisions for Multimedia and TCP Connections," in *Proceedings of IEEE VTC*, Sept. 1999.

[30] Y. Xiao, "Concatenation and Piggyback Mechanisms for the IEEE 802.11 MAC," in *Proceedings of IEEE WCNC*, Mar. 2004.

[31] L. Scalia, F. Soldo, and M. Gerla, "PiggyCode: a MAC Layer Network Coding Scheme to improve TCP Performance over Wireless Networks," in *Proceedings of IEEE GLOBECOM*, Nov. 2007.

[32] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," in *Proceedings of ACM SIGCOMM*, Aug. 1996.

[33] C. Perkins, E. Royer, and S. Das, "Ad-hoc On-demand Distance Vector (AODV) Routing," in *Proceedings of IEEE WMCSA*, Feb. 1999.

[34] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," *IETF RFC 2582*, Apr. 1999.

[35] B.-S. Kim, Y. Fang, T. F. Wong, and Y. Kwon, "Throughput Enhancement Through Dynamic Fragmentation in Wireless LANs," *IEEE Transaction on Vehicular Technology*, vol. 54, no. 4, pp. 1415–1425, July 2005.

[36] C. Chen, H. Luo, E. Seo, N. H. Vaidya, , and X. Wang, "Rate-adaptive Framing

for Interfered Wireless Networks," in *Proceedings of IEEE INFOCOM*, May 2007.

[37] M. Scharf, M. Necker, and B. Gloss, "The Sensitivity of TCP to Sudden Delay Variations in Mobile Networks," *Lecture Notes in Computer Science*, vol. 3042/2004, pp. 76–87, Apr. 2004.

[38] D. Qiao, S. Choi, and K. G. Shin, "Goodput Analysis and Link Adaptation for IEEE 802.11a Wireless LANs," *IEEE Transactions on Mobile Computing*, vol. 1, pp. 278–292, Oct. 2002.

[39] V. Erceg et al., "TGn Channel Models," *IEEE 802.11-03/940r4*, May 2004.

[40] M. Vutukuru, H. Balakrishnan, and K. Jamieson, "Cross-Layer Wireless Bit Rate Adaptation," in *Proceedings of ACM SIGCOMM*, Aug. 2009.

[41] S. H. Y. Wong, H. Yang, S. Lu, and V. Bharghavan, "Robust Rate Adaptation for 802.11 Wireless Networks," in *Proceedings of ACM MOBICOM*, Sept. 2006.

[42] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-friendly High-speed TCP Variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, July 2008.

[43] A. M. Guidotti, C. Raffaelli, and O. G. de Dios, "Effect of Burst Assembly on Synchronization of TCP Flows," in *Proceedings of IEEE BROADNETS*, Sept. 2007.

[44] D. Vardalis, "On the Efficiency and Fairness of TCP over Wired/Wireless Networks," Master's thesis, State University of New York at Stony Brook, 2001.

[45] S. H. Choi, "A Software Architecture for Cross-Layer Wireless Networks," Ph.D. dissertation, The University of Texas at Austin, May 2008.

[46] S. Hanemann, R. Jansen, and B. Freisleben, "Reducing Packet Transmissions in Ad Hoc Routing Protocols by Adaptive Neighbor Discovery," in *Proceedings of ICWN*, June 2003.

Vita

Won Soo Kim was born in Busan, Korea on May 13, 1975. He received his Bachelor of Engineering and Master of Engineering degrees in Radio Science and Engineering from Korea University in February 2001 and February 2003, respectively. From 2003 to 2005, he worked for Samsung Advanced Institute of Technology in Korea as a member of technical staff. In August 2005, he joined the Electrical and Computer Engineering at the University of Texas at Austin to pursue his doctoral degree. During his study, he was a member of the Wireless Networking and Communications Group.

Permanent Address: Daeseong Apt. 104-111 Mandeok 2-Dong Buk-Gu
Busan, 616-756, South Korea

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay, James A. Bednar, and Ayman El-Khashab.